5-2013

Reverse Engineering: WiMAX and IEEE 802.16e

Katherine Cameron
Clemson University, k_cam_11@hotmail.com

Reverse Engineering: WiMAX and IEEE 802.16e

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Electrical Engineering

---

by
Katherine C. Cameron
May 2013

Accepted by:
Dr. Richard Brooks, Committee Chair
Dr. Kuang-Ching Wang
Dr. Ian Walker

Abstract

Wireless communications is part of everyday life. As it is incorporated into new products and services, it brings additional security risks and requirements. A thorough understanding of wireless protocols is necessary for network administrators and manufacturers. Though most wireless protocols have strict standards, many parts of the hardware implementation may deviate from the standard and be proprietary. In these situations reverse engineering must be conducted to fully understand the strengths and vulnerabilities of the communication medium.

New 4G broadband wireless access protocols, including IEEE 802.16e and WiMAX, offer higher data rates and wider coverage than earlier 3G technologies. Many security vulnerabilities, including various Denial of Service (DoS) attacks, have been discovered in 3G protocols and the original IEEE 802.16 standard. Many of these vulnerabilities and new security flaws exist in the revised standard IEEE 802.16e. Most of the vulnerabilities already discovered allow for DoS attacks to be carried out on WiMAX networks. This study examines and analyzes a new DoS attack on IEEE 802.16e standard. We investigate how system parameters for the WiMAX Bandwidth Contention Resolution (BCR) process affect network vulnerability to DoS attacks. As this investigation developed and transitioned into analyzing hardware implementations, reverse engineering was needed to locate and modify the BCR system parameters.

Controlling the BCR system parameters in hardware is not a normal task. The protocol allows only the BS to set the system parameters. The BS gives one setting of the BCR system parameters to all WiMAX clients on the network and everyone is suppose to follow these settings. Our study looks at what happens if a set of users, attackers, do not follow the BS's settings and set their BCR system parameters independently. We hypothesize and analyze different techniques to do this in hardware with the goal being to replicate previous software simulations that looked at this behavior.

This document details our approaches to reverse engineer IEEE 802.16e and WiMAX. Additionally, we look at network security analysis and how to design experiments to reduce time and cost. Factorial experiment design and ANOVA analysis is the solution. In using these approaches, one can test multiple factors in parallel, producing robust, repeatable and statistically significant results. By treating all other parameters as noise when testing first order effects, second and third order effects can be analyzed with less significance. The details of this type of experimental design is given along with NS-2 simulations and hardware experiments that analyze the BCR system parameters. This purpose of this paper is to serve as guide for reverse engineering network protocols and conducting network experiments.

As wireless communication and network security become ubiquitous, the methods and techniques detailed in this study become increasingly important. This document can serve as a guide to reduce time and effort when reverse engineering other communication protocols and conducting network experiments.

Acknowledgments

This work would not have been possible without my advisor, Dr. Richard R. Brooks, who has provided guidance and encouragement throughout this study. I am grateful for his knowledge, mentoring, and help. Additionally, I would like to thank my committee members, Dr. Kuang-Ching Wang and Dr. Ian Walker, for their work in revising and editing the documentation and expertise and knowledge. Lastly, I would like to acknowledge the members of Dr. Brooks' research group for their help and encouragement throughout the process of completing and documenting this study.

Finally, I must acknowledge that this material is based upon work supported by, or in part by, National Science Foundation's grant, the EAGER-GENI Experiments on Network Security and Traffic Analysis CNS-1049765. Opinions expressed are those of the author and not the National Science Foundation.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This thesis documents our hardware investigation of the Bandwidth Contention Resolution (BCR) system parameters and their effect on the average throughput of other client WiMAX nodes when attackers maliciously set their own parameters. This includes documentation of reverse engineering IEEE 802.16e and WiMAX on a Linux machine. The following chapters document the steps we took. We hope that the details presented here can guide others doing similar wireless and network protocol investigations. The work began as an effort to conduct hardware experiments replicating the software work completed by Deng in [7] with hopes of verifying the results and/or providing an analysis of the differences between software simulations, specifically NS-2, and actual WiMAX hardware. The software simulations analyzed the ability of these parameters to be used as a Denial of Service attack on WiMAX client nodes. Over time, as the project evolved, the focus became a study of reverse engineering the hardware WiMAX protocol implementation.

Experimentation with hardware has proved challenging because a user is not supposed to change the system parameters and therefore the protocol does not allow for it. The BS is the entity that sets and controls the BCR system parameters. We have investigated various methods to modify the system parameters in hardware and this paper describes the steps taken in hopes of aiding others doing similar network research. Before presenting the details of these methods, an understanding of the importance of wireless communication protocols and wireless security will help one comprehend the motivation and significance of this work. The remainder of this section summarizes wireless communication, including its evolution to broadband wireless access and WiMAX, previous work conducted in network and WiMAX DoS attacks, and IEEE 802.16 security vulnerabilities.

1.1 Wireless Communication and Security

The idea of wireless communications, first formulated by Maxwell, was established at the end of the nineteenth century [35]. Analog microwave radios have been in use for many decades providing services such as television and mobile radios and, for the same amount of time, hobbyist have enjoyed the communication provided by HAM radios. But two technologies made wireless communications known by every household, cordless and cellular phones. Both of these were originally designed for low-data rates and high latency tolerance for voice. Unlike cordless phones, cellular technology needed to evolve to handle higher data rates and multiple uses. Cell phones, originally used only for voice, became a resource to connect to the web and access data available there. The need for faster and more reliable wireless protocols became apparent as users accessed increasing volumes of data, especially video streaming, online. Network bandwidth had to increase to satisfy user demand.

Today wireless communication is ubiquitous, provided by cell phones, computers, game consoles, and appliances. Bluetooth emerged as a leading protocol for short range, low data rate personal communications. IEEE 802.11 and WiFi, Wireless Fidelity Alliance, is used for local area networks with higher data rates that connect multiple devices within a building. Broadband wireless access (BWA) protocols cover large areas, typically a 5 mile radius, providing high data rates. BWA hird generation, 3G, technologies include GSM, UMTS, W-CDMA and are used currently in most smart phone applications. WiMAX, Worldwide Interoperability for Microwave Access, and LTE, long-term evolution, have emerged as competing 4G, fourth generation, technologies.

Wireless protocols operate on open channels since they are carried through air medium. This implies that anyone having means to receive and/or transmit at a specific frequency has access to the wireless channel. It is critical for security measures to be embedded into wireless protocols to protect the traffic confidentiality and integrity and prevent attacks [35]. Each wireless protocol has unique security issues. Though all are susceptible to man-in-the-middle and denial of service (DoS) attacks. WiMAX uses the Media Access Control layer (MAC) to implement and handle security. The MAC layer and security of WiMAX is discussed in following sections.

With the growing popularity of WiMAX and 4G technologies, our work originated from an interest in the security related issues of the WiMAX protocol. Many vulnerabilities of IEEE 802.16e had previously been investigated but there is little documentation about vulnerabilities in the system parameters of the Bandwidth Contention Resolution (BCR) process, a process commonly used in

WiMAX networks. A new question was proposed as to how these system parameters could be used to carry out a DoS attack. Deng conducted work in [2] using the NS-2 simulator to investigate this issue.

1.2 DoS and DDoS

Denial of Service (DoS) attacks and Distributed Denial of Service (DDoS) attacks have been used for a long time to block network resources. The large scale DDoS that caught the publics attention was in February 2000 and included attacks on major websites such as Yahoo!, Amazon, CNN.com, etc. The first reported large-scale DDoS attack using the Internet occurred at the University of Michigan in August 1999 and similar attacks had been targeting Internet Relay Chat (IRC) networks for many years prior [14]. At this time there was already a myriad of different DoS attacks that leverage weaknesses of common Internet protocols. Now with wireless networking protocols, there are only more DoS attacks that can be used to cripple a network. Most DoS and DDoS attacks comprise remote machines to use for attacks making it difficult to trace the attacker [27]. Though it is challenging to determine who initiates attacks, it is fairly easy to manually detect traditional DoS attacks using traffic capturing and/or monitoring. Clever DoS attacks are not only untraceable but are also arduous to detect.

A DoS attack is an attempt by a malicious user to prevent users of a service from being able to use or access resources. Common DoS attacks goals include [27]:

- Flooding a network, thereby preventing legitimate network traffic,
- Disrupting connections between two machines, thereby preventing access to a service,
- Preventing a specific individual from accessing a service,
- And disrupting service to a specific system or person.

DoS attacks can be broadly dividing into software and flooding attacks [21]. For wireless networks a third type is jamming. Flooding overwhelms the network and/or specific machines with traffic making services and/or transmission unusable. Common flooding attacks include User Datagram Protocol (UDP) flood, synchronization (SYN) flood, and smurf. In a UDP flood, return

addresses of UDP packets are spoofed to connect one machines UDP character generator to another machines UDP echo [14]. The SYN flood is for Transport Control Protocol (TCP) traffic. It spoofs the return addresses of SYN packets to a non-existing address and sends a number of these SYN packets to the target machine. The target machines transmission queue will be full of synchronization acknowledgment (SYN-ACK) packets destined for a machine that is not on the network and will end up waiting for an acknowledge (ACK) response that will never come. Lastly, a smurf attack leverages Internet Control Message Protocol (ICMP) by broadcasting ICMP ping requests with a spoofed return address of the target machine [14].

Unlike flooding which requires multiple malformed packets, software DoS attacks typically require only a few packets. Software attacks exploit bugs in the operating systems or applications. Many of these vulnerabilities and attacks can be found my searching MITREs Common Vulnerability and Exploit (CVE) database. Well known attacks include the ping of death and land attack [21]. In the first, the operating system crashes when a very large ICMP echo packet is received. The latter attack sends a single SYN packet with spoofed addresses of the target machine for both the source and destination fields corrupting the protocol stack.

Wireless networks are susceptible to flooding and software attacks. Additionally, each wireless protocol has unique communications that can be exploited for flooding and/or software DoS attacks on that specific protocol. For example, in IEEE 802.11 an attacker can spoof the MAC address in de-authentication and deassociation packets to disconnect target machines. In IEEE 802.16 range request message can be spoofed with the target machines MAC address requesting the lowest quality downlink burst profile. One more DoS attack that can be carried out on wireless networks is jamming. This technique transmits electromagnetic energy in frequency bands used by the wireless equipment to interfere with the wireless communications. There are various types of jammers including constant, deceptive, random, reactive, and intelligent [37].

Deng and Brooks investigated a new DDoS attack for WiMAX networks in [7] that manipulates the Bandwidth Contention Resolution parameters. This type of attack would appear random and would not result in sudden, drastic performance degradation making it hard to detect and differentiate from network noise. This study is a hardware extension of the work in [7] and is still investigating the possibility of using the parameters to carry out a DDoS attack. Next we will look at the security vulnerabilities of IEEE 802.16e and WiMAX including DoS vulnerabilities.

1.3 Previous Work

Many security flaws apparent in the original standard and additional security for support of mobility has been addressed by the privacy sublayer of IEEE 802.16e. This includes enhanced security, mobility management, and improved support for fast handovers. Issues still remain that leave the protocol vulnerable to Denial of Service (DoS) attacks hat can negatively affect network availability. Known security vulnerabilities include unprotected network entry initially, unencrypted management communications, no authentication on certain management frames, and sharing of keying materials in multicast and broadcast messages [31]. Numerous papers present, analyze, and simulate these vulnerabilities [1, 18, 26, 31, 43].

Naseer, Yoner, and Ahmed present an overview of security vulnerabilities found in IEEE 802.16e that can lead to DoS attacks in [31]. Some of the vulnerabilities stem from unprotected management communications and messages, including the:

- Ranging request (RNG-REQ),
- Ranging response (RNG-RSP),
- Fast Power Control (FPC), and
- Reset Command (RES-CMD).

Ranging occurs when subscriber station (SS) begins to acquire timing offset and power adjustment information from the BS to properly setup transmission. The RNG-REQ is the first message sent by the SS upon entering the network and is periodically sent afterwards to continue to keep transmission in alignment. A SSs also can use this message to inform the BS of their preferred downlink burst profile. RNG-REQ messages are neither encrypted nor verified, because they are sent before a Security Associate (SA), which would allow for encrypted traffic, is established. An attacker could intercept this message and send one requesting the least effective downlink burst profile.

Similarly, the RNG-RSP from the BS is not encrypted or authenticated. In this message the BS can set the SS of transmission power, change the uplink and/or downlink channel, terminate communications and re-initialization the MAC. A SS will accept any properly formatted RNG-RSP and change according to the details of the message. A malicious user could send a decoy RNG-RSP to set a SSs transmission power either very low [1] or high creating a water torture DoS attack

both having negative consequence SS. Lastly, they could change the downlink and/or uplink channel to different frequency range disturbing and breaking communications until the SS rescans to find correct frequency [31]. A DoS attack exploiting the RNG-RSP message is simulated using ns-2 network simulator in [1].

Other management messages of concern are the Fast Power Control (FPC), Mobile Neighbor Advertisement (MOB_NBRADV FPC), and Mobile Association Reply (MOB_ASC-REP). The FPC can be used similarly to the RNG-RSP message. Both the MOB_NBRADV FPC and MOB_ASC-REP messages can be used maliciously in networks that support handovers. Additionally, the authors in [31] introduce a new vulnerability related to Reset-Command (RES-CMD) message that forces a SS to re-initialize its MAC state machine. The purpose of doing this would be to reset a non-responsive or malfunctioning SS. Though this message is authenticated there are ways an attacker can cause a RES-CMD to be sent, temporarily disconnecting a SS from the network, and making it vulnerable to REG-REQ, REG-RSP attacks. An attacker that is part of network can receive UL-MAPS and then transmit during the given slots of other SS causing the transmission to be unintelligible. If they continue, the BS will assume that the victim is malfunctioning and will issue RES-CMD [31].

As seen in [20], authentication request messages can be used for DoS attacks by sending many request to the BS causing it to overload and become unable to serve other SS requests. Use of long keys further increases the computational load for authorization verification. The BS will not be able to defer between legitimate and illegitimate users till after processing the authentication request. Scrambling attacks, jamming only specific WiMAX connections and not the entire frequency spectrum, are another DoS attack that is a threat to all connection-based wireless protocols including WiMAX [4]. This type of attack may go unnoticed if carried out appropriately and is analyzed in [4] along with the solution of Dynamic CID Jumping Scheme (DCJS) proposed to defend against such an attack. Additionally, research in [1] examines whether vulnerabilities of IEEE 802.11 that enable DoS attacks (replay attack, MAC address spoofing, de-authentication, etc.) exist in the IEEE 802.16 standard. This study concludes that WiMAX is more robust but previously mention vulnerabilities unique to IEEE 802.16e still enable DoS.

Deng and Brooks proposed a novel DoS attack on WiMAX networks that manipulates the Bandwidth Contention Resolution (BCR) parameters. BCR is a process used in the Best Effort (BE) and Extended non-Real Time Polling Service (En-RTS) QoS classes to mitigate and handle

transmission contentions. In [7], the attack is presented and analyzed using software simulations on the NS-2 simulator. ANOVA analysis is applied to results to quantify the effect of the parameters on average throughput and packet-loss. Our current work is an extension of this study, which will be described further in Section 2.3.3. The original motivation was to analyze the possibility of using the BCR system parameters to conduct a DoS and determine the system parameter settings that most significantly affect a WiMAX client's vulnerability to such an attack. Some conclusions about these settings will be presented in Section 2.3.3 but due to difficulties with hardware experimentation we cannot make full conclusions to the best settings of the investigated system parameters.

Security is one clear reason for understanding the details of a communication protocol, but there are other important reasons as well. Knowing when and where bottlenecks occur, where data may be lost, and/or events that can cause network and/or resources to lock up can help eliminate many potential issues early on. A deep understanding of a network protocol and how it interacts with the hardware and equipment that use it provides great quality assurance. If you are to use a specific manufacturers wireless equipment, knowing this information will allow you to assure your own products and services. Unfortunately, many manufacturers make this information proprietary and/or obfusicated, leaving one with only the option of reverse engineering to gain this level of comprehension. Due to a lack of information about WiMAX's Bandwidth Contention Resolution system parameters handling by the WiMAX devices used in our hardware testbeds, this study has become a task of reverse engineering WiMAX hardware in search of these parameters.

1.4 Organization

The content of this thesis is outlined as follows. Chapter 1 describes the motivation behind the work by exploring the history, issues and challenges of wireless communication and WiMAX. Following Chapter 1, Chapter 2 introduces information necessary to understand this study and the methods analyzed for controlling the BCR system parameters in hardware. This includes details of WiMAX and the IEEE 802.16e protocol, and most importantly a detailed explanation of the BCR process of IEEE 802.16e. Additionally, an introduction to our previous NS-2 software simulations is presented providing more insight to the hardware phase of research that is detailed in this document. Lastly, an overview of the hardware resources and facilities used for our hardware WiMAX experimentation is given.

Chapter 3 presents a deeper analysis of the hardware testbeds used for WiMAX experimentation. This analysis also looks at the effect of two BCR system parameters on SS's throughput. Simulations were conducted in indoor and outdoor WiMAX environments. The results of these experiments is presented and a comparison is made between the two testbeds and previous software simulations. Following this chapter, the techniques investigated for modifying the BCR system parameters are presented.

Chapters 4, 5, and 6 outline our methods for conducting hardware experiments that replicate the NS-2 simulations. They look at how one might be able to control the system parameters separately from the BS. Many of the details and techniques used to reverse engineer WiMAX in search of the BCR system parameters are presented. We hope that these details will guide others carrying out WiMAX research and will provide a process for reverse engineering network protocols in hardware.

Chapter 4 examines Intel's Centrino Advanced + Wireless 6250 device and the open source drivers and software that are included with this device. Modifications made to the code for our study are presented here.

Chapter 5 looks at the Linux file system and memory. An abundant amount of information about WiMAX is stored within the file system and the memory in Linux. Are techniques for delving into and investigating this information in hopes of finding the BCR system parameters is presented.

Finally, Chapter 6 looks at the live WiMAX traffic captures that were made in hardware experimentation. This includes Ethernet traffic carried via WiMAX and a WiMAX management protocol that is used between the BS and ASN-GW. The later protocol is known as R6+ protocol and is a NEC proprietary version of the R6 protocol. We searched for the system parameters within the traffic captured and by doing so learned more about how Linux interfaces with WiMAX and the R6+ protocol. The purpose and results of the traffic captures is presented in this Chapter.

Lastly, in Chapter 7, a summary of our techniques is given with conclusions about what has been learned by this investigation. Other work that has been motivated by this study is briefly looked at as well, Section 7.2. Future work made possible by this study is presented last in Section 7.3.

Chapter 2

Background

The IEEE 802.16e standard and Worldwide Interoperability for Microwave Access (WiMAX) profiles include a broad range of technical specifications and procedures for handling various situations. The scope of the Background will be to present a brief overview of the IEEE 802.16e standard and WiMAX, highlight specific details of the IEEE 802.16e standard that we have attempted to leverage as a means to conduct our hardware experiments such as the MAC management messages that carry the BCR system parameters. As an example of the breadth of the IEEE 802.16e protocol, many important topics such as handovers, power management, bandwidth conservation, and adaptive antenna system will not be addressed throughout this study because they have no relation to the BCR system parameters. The security aspects of the MAC layer are presented here despite having no effect on the BCR process because it aids in understanding the previous WiMAX security research presented in Chapter 1.

Following the MAC layer information, we discuss the details of the BCR process. Originally, in [7] we propose that the BCR system parameters could be used to conduct a DoS attack on other WiMAX client SSs by not playing fair and following the designated system parameter settings. To understand why we hypothesized that these parameters could be used for a DoS attack, an in-depth understanding of the BCR system process is required and presented. Lastly, the past software simulations conducted in [7] is summarized and an overview is given of the hardware resource that are used for this investigation. The work in [7] was the catalyst for the current hardware phase of our study and a discussion of this work emphasizes our motivation for this study.

This chapter is structured as follows: we begin by covering the basic of the IEEE 802.16e

standard and WiMAX including the physical characteristics, the differences between the two, and the service oriented focus of QoS. Following the overview, detailed discussion of the MAC layer is given. Additional focus on the UCD, UL-MAP and TLV encoding is presented as subsections of the MAC layer. Next, the previous software simulations are illustrated. The experimental design and analysis used in both the previous software simulations and the hardware experiments, presented in Chapter 3, are explained in the beginning of Section 2.3 followed by the actual simulations. To conclude, the hardware resources used for this study are detailed. The GENI network and the ORBIT facilitie,s including the software and hardware elements that compose the ORBIT testbeds, are expounded.

2.1 WiMAX Overview

In broadband wireless access (BWA) there is a base station (BS) that controls access to a network for multiple subscriber stations (SS). Bandwidth is granted to SS by the BS using allocation schemes. Two-way communications is supported between SS and the BS. Communications are separated into a downlink channel, data sent from base station to subscriber station, and an uplink channel, data sent from subscriber station to base station. BWA uses point-to-multipoint or mesh access. The actual wireless broadband protocol used determines the technical requirements such as frequency, bandwidth, and operating range. WiMAX and IEEE 802.16e is a BWA solution that supports bandwidth rates up to 70 MBps and coverage radius of 5 miles on average. The original IEEE 802.16 standard was released in October 2001 and only supported line-of sight operations. It was later ratified in 802.16a, January 2003, and 802.16-2004, October 2004, to include non-line-of-sight frequencies, 2-11 GHz [25]. The standard was modified again in December 2005, 802.16e, to support mobility.

IEEE 802.16e-2005 standard operates in the 2-66 GHz frequency range. The use of scalable orthogonal frequency division multiple-access (SOFDMA) allows for scalable channel bandwidth from 1.25 to 20 MHz [36] and tolerance to multi-path fading. This multiplexing technique is based on orthogonal frequency division multiple-access (OFDMA) where bandwidth is subdivided into multiple frequency sub-carriers and allows for multiple access by multiplexing data streams from multiple users onto downlink and uplink sub-channels [8]. Figure 2.1 [8] shows the structure of an OFDMA frame including the use of symbols and sub-channels. The standard supports Time Division

Duplexing (TDD) and full and half-duplex Frequency Division Duplexing (FDD) and frame sizes between 2.5 and 20 ms. It also uses different modulation schemes to adapt to channel conditions, including binary phase shift keying (BPSK), quaternary PSK (QPSK), 16-quadrature amplitude modulation (QAM), and 64-QAM [32].

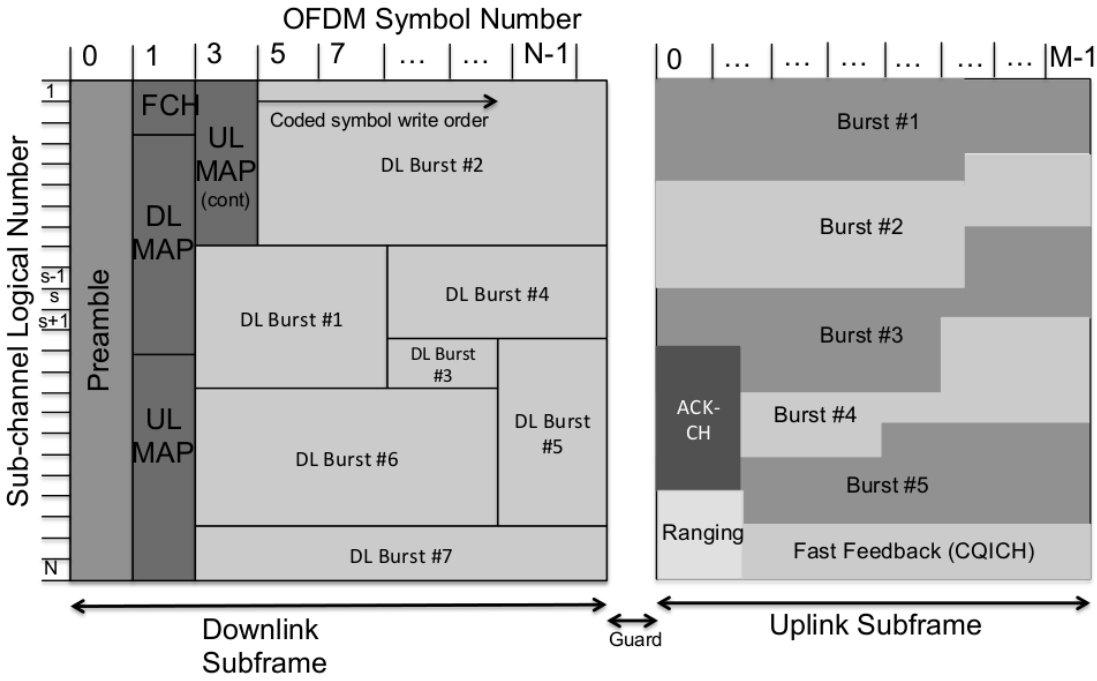

Figure 2.1: OFDMA Frame structure with sub-channels and symbols [8]

The WiMAX forum, which includes Airspan, Alcatel, Alvarion, Fujitsu, Intel, OFDM Forum, Proxim, and Siemens [36], was formed to determine an industry standard to promote equipment interoperability. The WiMAX forum serves the same purpose for the IEEE 802.16 standard as the Wireless Fidelity Alliance (Wi-Fi) does for IEEE 802.11. The industry standard includes specific profiles based on IEEE 802.16. For point-to-point and point-to-multipoint systems, Release-1 Mobile WiMAX profiles cover 5, 7, 8.75, and 10 MHz channel bandwidths for licensed worldwide spectrum allocations in the 2.3 GHz, 2.5 GHz, 3.3 GHz and 3.5 GHz frequency bands. The only frame size and duplexing currently supported in WiMAX profiles is 5 ms and TDD [8].

Additionally, IEEE 802.16e supports five quality-of-service (QoS) classes: Unsolicited Grant Service (UGS), real-time Polling Service (rtPS), non-real-time Polling Service (nrtPS), Best Effort (BE), and extended real-time Polling Service (ertPS) [26]. QoS is used to flexibly support simultaneous use of a diverse set of IP services [9]. All classes other than UGS use the bandwidth request

procedure via the uplink channel for SSs to request access to the network. Contentions, data collisions on the network, are a problem in the uplink channel during bandwidth requests and are not an issue in the downlink channel [25]. There are two mechanism used in WiMAX to handle request contentions: centralized polling and contention-based random access [32]. The service class determines whether contention-based or polling is used. Our work focuses on the contention-based bandwidth request process known as Bandwidth Contention Resolution used in both BE and nrtPS QoS classes.

The BE class is used by most Internet applications [26] and other application that require no minimum data rate to properly function. Fore example, when the network is congested, BE nodes typically cannot transmit packets until conditions improve. In BE, SS transmission requests may be made via the BCR process or unicast polling. Though the standard does not mandate the BS to provide unicast request opportunities [35]. The other QoS class, nrtPS, is best suited for variable sized, delay tolerant data that requires a minimum data rate for functionality. The use of periodic unicast request polling at interval of 1s or less provides the minimum data rate. Additionally, nrtPS nodes can use the BCR process to request transmission. This class suits applications such as File Transfer Protocl (FTP). Our work in finding and analyzing the BCR system parameters in hardware is important for applications that use both these QoS classes.

2.1.1 MAC Layer

IEEE 802.16 uses the Open Systems Interconnection, OSI, layer model, Figure 2.2. The protocol defines only the two lowest layers [35]:

- Physical link, layer 1 and
- Data link, layer 2.

The Media Access Control (MAC) layer of IEEE 802.16 is the core of the data link layer and establishes and manages connections between the BS and SSs. This layer is connection-oriented, all services map to connections that are assigned unique connection IDs (CIDs) [35]. The MAC layer allows for dynamic resource allocation allowing the connection to adapt based on data or transport needs. It is based on the DOCSIS standard and capable of supporting various types of data simultaneously [8]. This layer can be subdivided into three sublayers: Convergence Sublayer (CS), Common Part Sublayer (CPS), and the Security Sublayer.

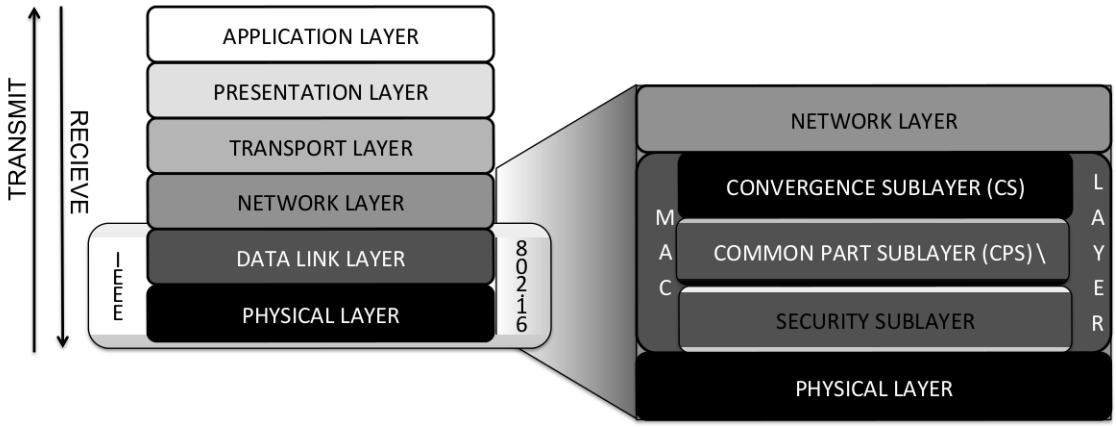

Figure 2.2: IEEE 802.16 and the OSI Model

The convergence sublayer interfaces with upper layers of the OSI model and prepares information for MAC layer handling when transmitting data and strips remaining MAC layer information when receiving. Essentially it accepts traffic from other networking protocols such as Asynchronous Transfer Mode (ATM), IEEE 802.3 (Ethernet), Internet protocol (IP), point-to-point protocol (PPP), etc. [35]. Separate parts of the CS handle ATM and packet-based traffic. In between the physical layer and CS MAC layer, lays the CPS.

The CPS is the heart of the MAC standard. In this sublayer, bandwidth allocation, connection establishment, and maintenance of the connection is handled. This includes scheduling, QoS management, resource radio management (RRM), and frame construction. There are many defined MAC management messages that allow the CPS to communicate with SS. Most of these messages were defined in the IEEE 802.16-2004 standard [16] with a few additional messages for handling mobility defined later in the IEEE-802.16-e standard. A list of these management messages can be found in Appendix A.

During initialization, the CPS establishs three types of management connections between the BS and SS prior to any data connections [39]. These connections include the basic, primary management, and secondary management connections. The basic connection is not encrypted and is for delay sensitive MAC management messages. The primary connection is also for MAC management messages but handles messages that are time tolerant. Lastly, the secondary management connection uses IP datagrams and carries standard-based messages, DHCP, TFTP, SNMP, etc. Additionally, there are specific CIDs for initial ranging, broadcast and multicast messages [39]. The

initial ranging CID is 0x0000 and broadcast CID is 0xFFFF, all other CIDs are assigned in between these two special case CIDs. Our work is particularly concerned with the broadcast connection, 0xFFFF, because the BCR system parameters are broadcast to SSs. The following section will detail the broadcast message that carries them.

The security sublayer is a critical element of the MAC. This layer is an improvement from the MAC security sublayer of 802.11 [25] with many added security features to protect the integrity of the SS and the network addressing flaws in the previous wireless standard. The security for mobile WiMAX networks as described in [8] requires the following:

- Strong mutual device authentication,
- All commonly deployed authentication mechanisms based on consistent and extensible authentication framework,
- Data integrity, replay protections, confidentiality and non-repudiation with applicable key lengths,
- Use of MS initiated security measures such as VPNs, and
- Standard secure IP address management.

To meet the above requirement, this sublayer can be divided into two main components: data encapsulation and key management [35]. Data encapsulation is used to secure data packets as they travel over the network. This component includes multiple encryption and authentication algorithms and rules for applying them to data payloads. Key management implements Privacy Key Management (PKM) protocol to safely exchange keys for authentication and, additionally, control of access to certain services. The IEEE 802.16e security specifications have been improved from IEEE 802-16-2004 to better secure WiMAX and address further security issues of mobile nodes. For example, the 2004 standard used PKMv.1, which uses one-way authentication, only authenticating the SSs. IEEE 802.16e improved security by using PKMv.2, which supports mutual authentication using certificates to authenticate both SS and BS. Also, as part of the IEEE 802.16 standard, security associations (SA) are defined and used. An SA is the set of information that is shared between the BS and SS to support secure communications. The SA can be shared between the BS and a single SS or between the BS and multiple SSs. The SA establishes the Cryptographic Suite (the type of

encryption, authentication, TEK exchange) that will be used. A summary of the various security protocols implemented in IEEE 802.16 and 802.16e is presented in Figure 2.3.

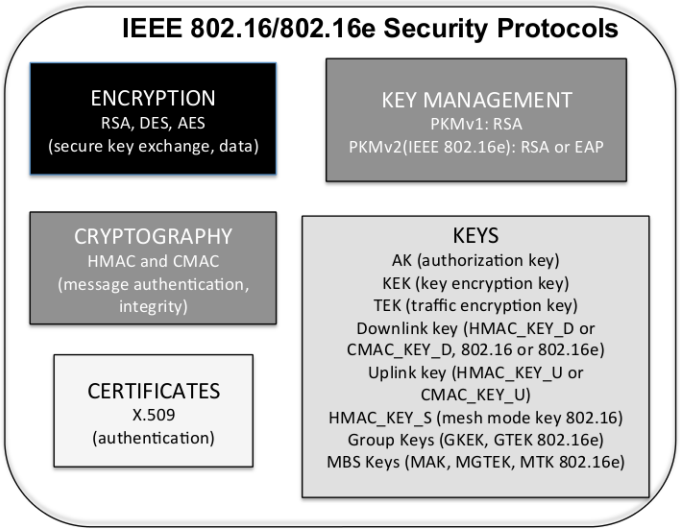

Figure 2.3: IEEE 802.16 and 802.16e security protocol block diagram

Lastly, the physical layer specifies the configuration and operation for the physical medium. This includes signal type, modulation and demodulation, transmission power, etc [35]. Many of these requirements are detailed in Section 2.1. The physical layer is one option that would allow us to change the BCR system parameters, if the correct equipment and facilities were available. This requires constructing and transmitting a spoofed UCD message with the correct BSID and the desired system parameters. We do not have the ability to use such resources for this study. Instead we are looking at MAC management messages and man-in-the-middle attacks on the messages. This will be described in Section 6.3.

2.1.2 Uplink: UCD and UL-MAP

The MAC messages that are particularly important to our work are the uplink map, UL-MAP, and uplink channel descriptor, UCD. These two management messages carry a variety of information for SSs, including system parameters, transmission request opportunities, and transmission grants. The BS transmits them in the downlink subframes periodically [35]. As pictured in Figure 2.4, the UL-MAP follows either the downlink map (DL-MAP), if one exists, or the frame control header (FCH). If the UCD message exists in the subframe, it will follow the UL-MAP pre-

ceded by a DCD, if there is one. These messages are included with the first downlink burst following the FCH.

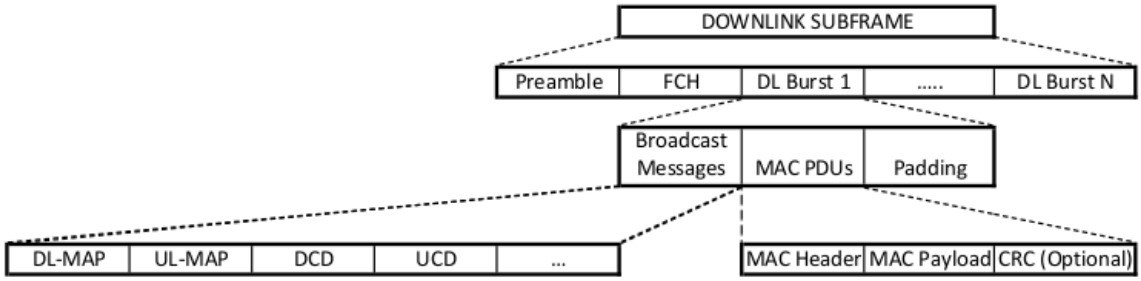

Figure 2.4: WiMAX downlink channel

The UL-MAP tells each SS when they can transmit and when transmission request opportunities are available. UL-MAP's contain the following:

- Uplink Channel ID, identifier of the uplink channel that message refers to,
- UCD Count, is equal to value of Configuration Change Count of UCD (UCD describes the uplink burst profiles for the specific UL-MAP),
- Allocation Start Time, effective start time of uplink allocation specified in UL-MAP, and
- Map Information Elements (IEs), contents dependent on PHY specification but defines uplink bandwidth allocation.

For the Map IEs, each uplink bandwidth allocation starts at the given offset relative to allocation start time and is specified in physical slot (PS) units [16]. The PS is dependent on the physical layer. The Uplink Interval Usage Code (UIUC) indicates the burst profile that should be used for each allocation. The UIUCs are defined in the UCD messages. The UCD Count parameter assures that the correct UCD message is used to translate the UIUC into burst profiles. A visualization of UL-MAP message can be seen in Figure 2.5.

As seen in the UL-MAP, the UCD defines the characteristics of an uplink physical channel with the UIUC. The maximum allowed period for UCD transmission is 10 s [16]. This message carries system parameters used in contention resolution for SSs as well as uplink burst profiles. The following parameters must be included in the UCD:

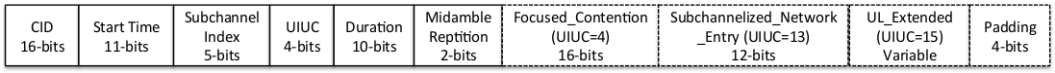

Figure 2.5: UL-MAP IE construction

- Configuration Change Count: incremented each time the values of channel descriptor change. Allows for SS to quickly determine whether or not to process rest of descriptor. [16]
- Ranging Backoff Start: initial backoff window size for initial ranging, expressed as power of 2. Can be set between 0-15. Most significant bits (MSB) not used.
- Ranging Backoff End: final backoff window size for initial ranging, expressed as power of 2. Can be set between 0-15. MSB not used.
- Request Backoff Start: initial backoff window size for bandwidth requests for contention resolution, expressed as power of 2. Can be set between 0-15. MSB not used.
- Request Backoff End: final backoff window size for bandwidth requests for contention resolution, expressed as power of 2. Can be set between 0-15. MSB not used. [16]
- Contention-based reservation timeout: time that SS waits to see transmission grant before sending another transmission request
- Bandwidth request opportunity size: size in PS of physical burst an SS may use to transmit Ranging Request message in contention ranging request

Additionally, for this study the ranging process in not examined. This leaves four important parameters defined in the UCD message for our study: request backoff end, request backoff start, contention-based reservation timeout, and bandwidth request opportunity size. All four of these parameters are used in the Bandwidth Contention Resolution.

The UCD also defines the profiles of the UIUC. These definitions are TLV encoded. This type of encoding will be covered in Section 2.1.3. There are 20 available modulation and coding schemes for uplink burst profiles [35]. The format of the UCD message is pictured in Figure 2.6.

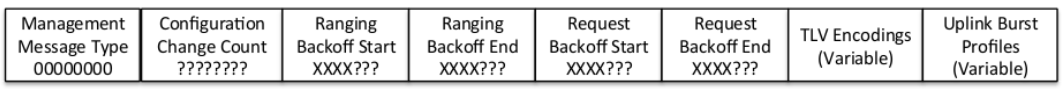

Figure 2.6: UCD message construction

Another important aspect of the UL-MAP and UCD message is they are needed to keep a SS locked on to a specific uplink channel. An SS is considered to have valid uplink parameters as long as it continues to successfully receive the UL-MAP and UCD messages. If at least one of these is not received within 50s, the SS will not use that uplink channel. The standard specifies the time limit by equation 2.1. The UCD maximum interval is 10s leading to a maximum time limit for receiving a UL-MAP or UCD of 50 seconds.

$$[\text{max time interval ULMAP or UCD}] = 5 * [\text{UCD Interval Maximum}] \tag{2.1}$$

We will see later why this information is important. Section 7.3 describes a technique for possibly temporary manipulating the BCR system parameters by taking advantage of this time interval.

2.1.3 TLV

MAC messages in IEEE 802.16e use type, length, value (TLV) encoding. This encoding scheme is a way of storing data to facilitate quick parsing and is a type of Basic Encoding Rules (BER) that is defined by the International Telecommunication Unions Telecommunication Standardization Sector (ITU-T) X.690 standard [23] for Abstract Syntax Notation One (ASN.1) encoding. For each parameter encoded as TLV, the first byte identifies the parameter type, the following byte(s) are indicative of the total length in bytes of the value field, and the last are the parameter value [19].

The size of the Length field follows the definite form of the X.690 standard [16]. The definite form defines the length of the Length field as 1 byte if the Value field is less than 127 bytes. The MSB of the single byte is set to 0 and the seven least significant bits (LSB) indicate total length in bytes of Value field. On the other hand, if the length of Value field is more than 127 bytes, the Length field is one byte larger than the needed bytes to indicate the correct length. The MSB of the first byte is set to 1 with the remaining LSB specifying the remaining number of bytes of the Length field. The remaining bytes denote the total length in bytes of the Value Field.

The parameter type indicates the encoding rules that are used. IEEE 802.16 uses the same type for various messages, but within the message unique TLV types are used. The system parameters encoded within these messages are different and the system knows which it is currently handling, therefore it can determine the explicit encoding used. For parameters that are TLV

encoded and are referenced by multiple message types, a globally unique TLV type is used to guarantee uniqueness. Examples of such parameters are transmit power, service flow descriptors, and security information.

TLV encoding is commonly used for physical layer specification in IEEE-802.16 MAC management messages. These types of encoding begin at type 150 and go up. Non physical layer encodings begin at type 1. This scheme is used in the UCD message to encode the UCID burst profiles and the Radio Resource Management (RRM) messages. Additionally, it is used for configuration parameters such as software update, hardware version, DHCP, etc., and other MAC management messages but the UCD and RRM messages are the only TLV encodings relevant to this research.

In the UCD message, contention-based reservation timeout, bandwidth request opportunity size, ranging request opportunity size and frequency is encoded along with burst profile information, ranging information, and bandwidth request codes. The BCR system parameters are TLV types 24 for contention-based reservation timeout, bandwidth request opportunity size and ranging request opportunity size, respectively. Frequency is encoded as TLV type 5. Types 150 -172 are used to encode OFDMA settings. For the burst profiles of OFDMA, encoding types 150 - 152 are used again. The code type and modulation are type 150 and the Value field from 0 - 25 represents twenty-six different modulation schemes with 26-255 being reserved. This encoding can be found in Section 11 of the IEEE 802.16-2004 standard [16].

IEEE 802.16e defined new TLV encodings in addition to encoding defined in IEEE 802.16-2004. New types include RRM messages and Full UCD Setting. More information about these two encoding types can be found in Table 2.1 and 2.2 [10]. Further detail of Full UCD Setting encoding can be found in [17].

The UCD message alone, separate from the RRM message, contains TLV encoded parameters as well. It was originally hypothesized that we may need to search and/or decode the encoded information to determine with certainty that they encapsulate the BCR system parameters, and this is the purpose of presenting the information in this section. Upon further investigation using the techniques detailed in Chapters 4, 5, and 6, we did not find any indication of the BCR system parameters that required further analysis of items such as TLV encoded parameters.

Table 2.1: IEEE 802.16e new TLV type, RRM BS Info

RRM BS INFO	
Type	159
Length	Variable
Value	Compound
Description	Contains a description of BS parameters which are not related to a specific MS.
Elements (Sub-TLVs)	
TLV Name	BS ID
	Available Radio Resource DL
	Total slots DL
	Available Radio Resource UL
	Total slots UL
	Radio Resource Fluctuation
	DCD/UCD Configuration Change Count
	DCD Setting
	UCD Setting
	Full DCD Setting
	Full UCD Setting
Use This TLV	RRM Spare_Capacity_Rpt, RRM Neighbor_BS_Resource_Status_Update, RRM Radio_Config_Update_Rpt.

Table 2.2: IEEE 802.16e new TLV type, Full UCD Setting

FULL UCD SETTING	
Type	73
Length	Variable
Value	Compound
Description	This is an IEEE802.16e-2005 defined TLV. The UCD_settings is a TLV value that encapsulates a UCD message (excluding the generic MAC header and CRC) that may be transmitted in the advertised BS downlink channel. This information is intended to enable fast synchronization of the MS with the advertised BS downlink.
Parent TLV(s)	RRM BS Info

2.2 Bandwidth Contention Resolution

All of our work revolves around the Bandwidth Contention Resolution (BCR) process and the system parameters used to control the execution of this process. The work in [7] used the NS-2 simulator to analyze the effects on average throughput and packet-loss rate when a set of attacker SS's do not follow the BS's settings of the BCR system parameters. Our study desires to take the analysis one step further and conduct the same experiments in hardware. This is a challenging task since the protocol and hardware implementations do not directly allow for such behavior. An explanation of the BCR process will clarify two things: why manipulating the system parameters could adversely affect other users on the network and why the BS controls the values of them.

In the Bandwidth Contention Resolution process, a SS has system parameters that are set periodically by a broadcast message from the BS known as the Uplink Channel Descriptor (UCD). These parameters are initially acquired during SS initialization as the second step after finding a downlink channel and synchronizing with the BS [16]. The steps during initialization are carried out in the following order:

1. Scan for downlink channel and synchronize with BS,
2. Obtain transmit parameters from UCD message,
3. Perform ranging,
4. Negotiate capabilities,
5. Authorization and key exchange,
6. Registration,
7. IP connectivity,
8. Time of day,
9. Operational parameters, and
10. Set up connection.

The UCD defines characteristics of the uplink channel and burst profiles. The BS tracks multiple aspects of the network including number of users and channel conditions. The BS selects

uplink characteristics and burst profiles that provide the best and/or needed performance required by each SS. The BS also modifies these parameters as network conditions vary to maintain the performance. Some of the parameters included in UCD and set by the BS are request backoff start, request backoff end, ranging backoff start, ranging backoff end, contention-based reservation timeout and bandwidth request opportunity size. When a SS needs bandwidth allocation it must request it from the BS, and these previous parameters control the binary truncated exponential backoff algorithms used in Bandwidth Contention Resolution. This process handles the issue of two SS transmitting information at the same time and/or SS transmitted data not being received by the BS. It is a means of handling the potential collisions on the network.

In BCR, when a SS has data they want to transmit they randomly choose a number in the range of [0, *backoff_start* -1] and defers a number of contention request transmission opportunities equal to the randomly selected value before transmitting their request. The contention request transmission opportunities are defined in the UL-MAP, uplink map, which also defines bandwidth allocation for SSs. The UL-MAP, like the UCD message, is sent by the BS periodically to all SSs. After sending a request, the SS checks each UL-MAP message for a pre-defined time period that is determined by contention-based reservation timeout parameter to check if transmission was granted. If no data grant is seen, the SS will increase *backoff_start* by one, doubling the range for the randomly selected variable [19] and repeat the process. This continues until *backoff_start* equals *backoff_end*, at which the protocol data unit (PDU) is dropped and the process restarts with the next PDU [39].

There are two situations that can ultimately stop the bandwidth contention process. First, transmission opportunity is granted. Second, the number of retries executed by the SS is equal to *request_retry*. If transmission opportunity is not granted before one of these scenarios occur, the request is considered denied and the data is dropped. With an understanding how each system parameter affects the contention process, hopefully it is clear how they can be manipulated to limit transmission opportunities of other SS. For example, if one was to set *backoff_start* very low, say 1, then they will always be able to send there transmission request in the first or second contention request transmission opportunity. If this SS continued to send requests in could limit the ability of other SS to transmit there requests. Next we will look at software simulations that actual analyzed this particular system parameters as well as others and discuss the results.

2.3 NS-2 Software Simulations

Deng and Brooks conducted software simulations that analyzed the BCR system parameter effects on SS average throughput and packet-loss in [7]. These simulations were conducted with the network simulator, NS-2, of the Information Sciences Institute. The NS-2 is a discrete event simulator targeted at networking research with support for TCP, routing, and multicast protocols over wired and wireless networks [22]. Under the guidance of Brooks, this study has attempted to replicate these simulations in hardware. The following section will summarize the NS-2 simulations and results while explaining the experimental design and analysis used. This design and analysis is also used for the hardware experiments detailed in Section 3.4.

2.3.1 Factorial Experimental Design

Factorial experimental design is a technique that produces substantial, reliable information from the a minimal set of data and minimum number of test trials. Data for all combination of factors is collected. This produces robust results and significant information about individual parameters and parameter interactions. With this method, the tests for any one factor setting contains values collected for all settings of other factors, allowing higher certainty that obtained results are due only to specific factor settings rather than other varying factors that are treated as noise. This provides significant information about the parameter and higher certainty of effects it produces. Furthermore, the ability to analyze the effect of parameter interaction on a result is beneficial as well.

The operating characteristic curves of type II error probability [30] are used to determine when to reject and accept a null hypotheses. The characteristic curves model the extent to which the null hypothesis is false for a statistical test of a fixed sample size. For this study, the null hypothesis is a parameter's settings have equally no effect on the result. The alternative hypothesis is that at least one of the settings does have an effect on the result. Statistical tests are used to quantify and validate the hypothesis as we will see in Section 2.3.2.

We construct our experimental design so that the null hypothesis should be rejected with probability of 95% producing a significance level, α, of 5%, if the difference in average throughput of all SSs between two values of a parameter is as great as 5000. We assume standard deviation of throughput for all SSs is 2000 bps. We require, a confidence of 99% or greater, equivalently a type II error, β, less than 1%.

In experimental design, determining the correct number of replications is critical. Using number of factors, X, levels for each factor, Y, the desired β, and *Chart V. Operating Characteristic Curves for Fixed Effects Model Analysis of Variance* [30] one can find ϕ. With standard deviation and maximum difference between parameter values, the following equation, 2.3.1, can be used to determine number of replications, n, to assure β.

$$\phi^2 = \frac{n(Y^(X-1))(5000^2)}{XY(2000^2)} \tag{2.2}$$

2.3.2 ANOVA Analysis

Analysis of variance (ANOVA) [42] was used on simulation results to isolate which parameters and/or parameter combinations affect SS throughput and vulnerability to DoS most significantly. Multiple factor ANOVA analyzed the 1st, 2nd and 3rd order parameter effects on network performance. First order effects quantify how much an individual parameter alters the outcome, and second and third order effects quantify additional influence of interactions of two and three parameters respectively.

To understand ANOVA, allow response Y to be affected by two parameters A and B. A has a treatments and B has b. Two-way ANOVA compares the effects of treatments of each parameter and the parameter interaction. It tests the equality of treatments of A, different a's, equality of treatments of B, different b's, and whether A and B interact, by finding the sum of squares (SoS), mean square (MS), and applying the F-test to calculate F0, the F statistic. The variable F0 follows an F-distribution and the questions is if it follows the same F-distribution as the entire data set. The F-test finds the distribution variance of one specific parameter, in this case A, B, or interaction of A and B. This variance is then compared to the distribution of variance for the entire data set to determine if they are the equivalent. If they are, than the parameter's settings are not significant on data. One can consider F0 is the probability that a random process could produce a value at least as extreme as the observed value.

To determine whether or not to reject the null hypothesis one first finds F_α, the value for the desired significance level from the F-distribution that fits the entire sample set. This F-distribution that is appropriate is different depending on whether we are analyzing first, second, or third order

interactions. Next, F0 is calculated based on the the variance for the specific parameter's subset of data. If F0 is greater than F_α that parameter is significant. A correlation to F0 is the *p-value*. The *p-value* is the probability that an F statistic is greater than F0, $P(F > F0)$. If the *p-value* is less than α than the parameters is significant.

To more intuitively quantify the amount of variance seen R-SQUARE is used. R-SQUARE is commonly used in data analysis and interpreted as percent of variance in data explained by a parameter, X [30]. Factor X can be a parameter or parameter interaction. The equation for R-SQUARE is presented in 2.3.2. For R-SQUARE, normal distribution of error is assumed. The experimental design confounds readings from multiple factors so when a parameter significantly affects the result, it is likely that the values for other factors do not follow normal distribution. This may lower the significance of the test [30], but the F-test, which we use to determine whether or not a parameter is significant initially, is robust to the normality assumption and only slightly affected.

$$R^2 = \frac{SoS_X}{SoS_{total}} \tag{2.3}$$

Two-way ANOVA is used to analyze average throughput for each SS overall replications. For first order interactions, the null hypothesis can be summarized as the average throughput for SSs for different treatments of a parameter are equal. For second and third order interactions, the null hypothesis is that no parameter interaction exists. We still use the same criteria earlier stated to determine whether to reject or accept the null hypothesis. The hypothesis being rejected indicates specific treatment of the parameter has significant affect on throughput or parameter interaction occurs, the alternative hypothesis. Using these statistical tests, an abundant amount of information about the effect of parameters is derived from the data.

2.3.3 Software Simulations

Unlike past work in WiMAX security vulnerabilities, [1,26,31,43], this research investigates how DoS attacks can be conducted using WiMAX system parameter settings. The parameters examined are *bw_backoff_start*, *bw_request_retry*, and *frame duration*. Additionally, one other parameter is considered: attacker to user ratio (number_of_attacker/user). The Bandwidth Contention Resolution process, as explained in Section 2.2, uses *bw_backoff_start*, *bw_backoff_end*, and *bw_request_retry*

to determine the delay an SS waits in between transmission requests and the number of times that an SS will transmit bandwidth requests before dropping data. If an attacker station ignores the BS mandated setting for these values they can potentially give themselves priority for sending transmission requests which results in more granted opportunities to transmit data. This can adversely affect regular user SS by limiting their opportunities to either transmit their bandwidth requests and/or data.

Software simulations were conducted using the NS-2 simulator [22] by Deng [7] to examine the effect of setting each of these parameters on client and attacker SSs to a low, medium, and high value. Please see Table 1 for parameter settings, the settings that begin with dos refer to attacker SSs where as settings that begin with bw refer to client SSs. The NS-2 simulator was configured to imitate a WiMAX network that has a single BS, centrally located, and 100 SSs, which include both client SSs and attacker SSs, placed along a 150 meter radius of the BS. Connected through a wired interface to the BS is a sink node that acts as a receiver for all traffic. The bandwidth was set to 10 MHz and each SS used a UDP constant bit rate traffic generator to send 1492 B packets to the sink node at intervals of 0.5 s. The system was configured in such a way to avoid packet dropping.

Table 2.3: Software simulation parameters

Parameter	**Treatment 1**	**Treatment 2**	**Treatment 3**
frame duration (s)	0.004	0.01	0.02
number of attacker/user	20/80	50/50	80/20
dos_backoff_start	1	3	5
dos_request_retry	2	6	10
bw_backoff_start	1	3	5
bw_request_retry	2	6	10

Seven replications of each parameter setting combination were conducted for a total of 5103 experiments. Throughput and packet-loss measurements were collected for each replication, and ANOVA [42], analysis of variance, was used to determine how these parameters and the parameter interactions affected throughput. The results of these simulations and further explanation of the ANOVA analysis can be found in [7]. Figure 2.7 is a visual representation of the results. In this image, we see that 31% of the variance in throughput is caused by frame duration, followed by 22% from the client SSs request retry. Lastly, the client SSs backoff start is responsible for over 5% of

variance and taking the combination of these three parameters and their interactions accounts for 86% of the variance in average throughput.

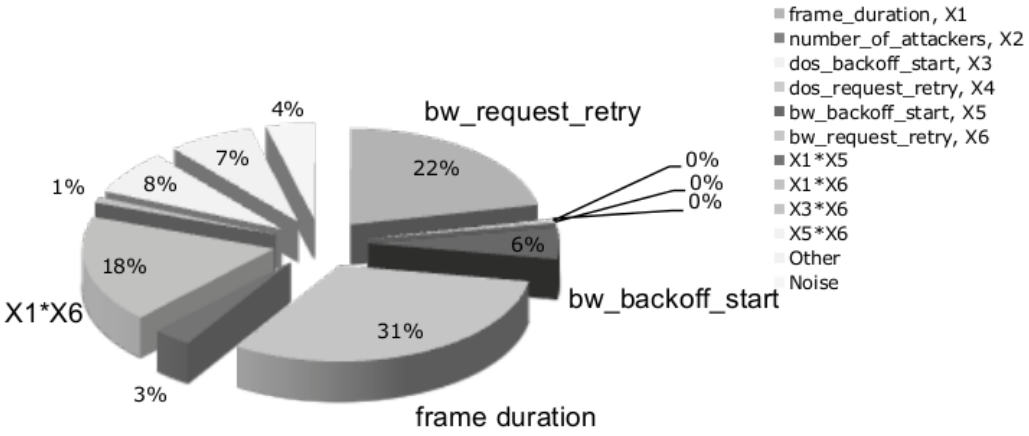

Figure 2.7: Piechart of ns-2 results for average throughput

ANOVA analysis indicates that frame duration and *bw_request_retry* and their combination affect throughput most dramatically. The affects of other parameters are minor in comparison. Other conclusions that can be made include:

- Throughput decreases drastically when frame duration is set to a value of 2 ms.
- Throughput increases drastically when *bw_request_retry* is increased from 2 to 6.
- The ratio of attackers to users does not significantly affect the impact of DoS attacks.

Additionally, one last point to highlight is that it is the client SS's settings of *request_retry* and *backoff_start* that contributes greatly to the variance seen in average throughput. This suggest that the client's settings of the system parameters is what matters most in decreasing the vulnerability to an attack that manipulates the BCR system parameters. The analysis indicates that a high level setting of *bw_request_retry* provides the best throughput. The other two client settings, *frame duration* and *bw_backoff_start* both should be set to low values to provide robustness to a DoS attack of this type. The best settings for these are not surprising and are easily explained by the BCR process. The high setting of *bw_request_retry* allows a SS to complete multiple attempts to transmit providing more robustness if there is disturbances on the network. The low setting of *bw_backoff_start* produces a shorter wait time for sending the a transmission request, and ultimately

if the request fails, produces a shorter time till the next attempt. Lastly, a small frame duration allows more packets to be sent producing a better probability for some of the packets to get through.

2.4 GENI and ORBIT

GENI, global environment for network innovation, is a virtual laboratory for at-scale networking experimentation supported by the National Science Foundation. The backbone network is constructed of National Lambda Rail (NLR) [34] and Internet2 [24] links creating a high capacity backbone to support multiple GRE tunnels for experimentation. The main object of this virtual laboratory is to allow researchers to explore future Internet technology. Many other applications have become possibly with such a large-scale network for experimentation such as wireless, sensor networks, and security research.

ORBIT, open-access research testbed for next-generation wireless networks [45], is one entity of the GENI network. ORBIT is ran by Rutger University's WINLAB and provides eleven different software-defined radio testbeds. The resources are remotely accessible, completely configurable, support experimental scalability and reproducibility, and extensive measurement. There are two integral parts that allow this, the ORBIT management framework (OMF) [40] and ORBIT measurement library (OML) [44]. Both of these services are stored on a single server known as the OMF/OML server. OMF includes software and hardware that allows an experimenter to write an experiment in a high-level language and then translate it to control the resources to carry out the experiment. OMF handles the translation and control of execution of the experiment.

2.4.1 OMF and OML

The basic OMF framework includes the following software and hardware:

- Support servers, repositories,
- Collection server (CS),
- Disk-loading service,
- Node handler,
- Node agent,

- Experiment controller service, and
- Chasis manager (CM)/ Aggregate manager service.

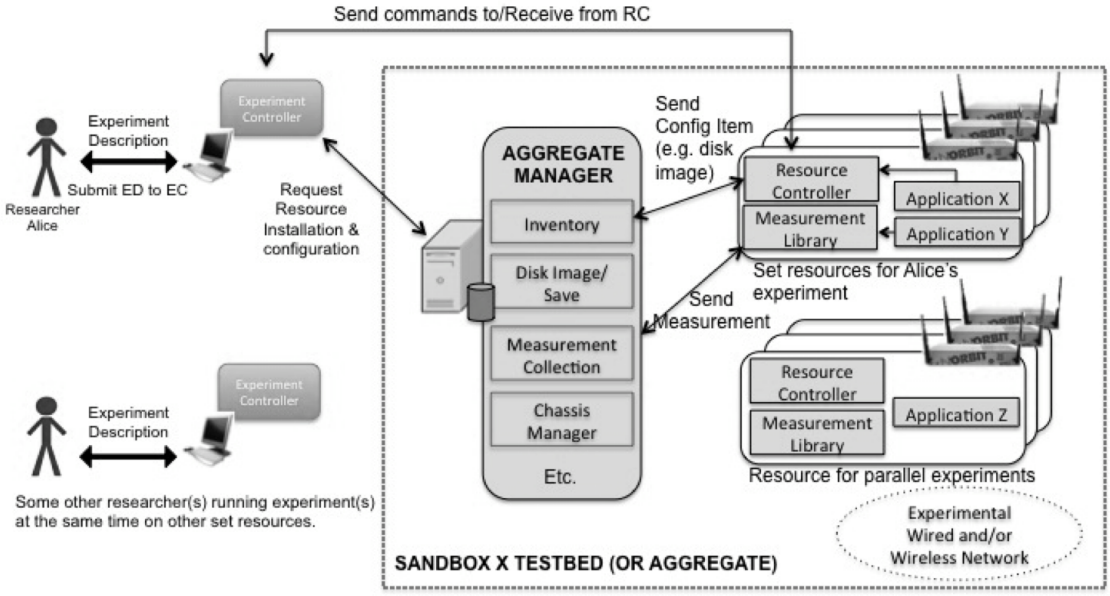

Figure 2.8: ORBIT Management Framework [33]

A block diagram of the ORBIT framework can be seen in Figure 2.8 [33]. Support servers compromise servers for web services and experimentation and data storage. The latter servers are called repositories in the ORBIT system. Predefined applications such as traffic generators and traffic receivers, i.e. ORBIT traffic generator (OTG) and ORBIT traffic receiver (OTR), have been developed and are stored in the repository for experimenter use. The collection server stores measurement data using an SQLite database. The node handler and node agent work together to control the experiment via the Experiment Controller Service. The disk-loading service enables quick re-imaging of hard disks on individual nodes [40]. The node agent is on the end nodes and the node handler, stored on the server, sends experiment scripts to each node via multicast messages. Lastly, experiment controller service and the aggregate manager services are stored on the OMF/OML server. The experiment controller, as mentioned before, coordinates the node handler and node agent. The CM has a dedicated Ethernet connection to each node that allows it to monitor status of hardware and remotely reset and power on and off each node.

OML enables measurements to be easily integrated into the user experiment scripts. It is a distributed client-server software framework that enables real-time collection of data from the

various applications being executed during an experiment [45]. The library defines data structures and functions for making and handling measurements in XML data reduced (XDR) format. This includes transmitting and receiving and encoding and decoding measurements [44]. OML offers measurement metrics and various filters that can be applied to these metrics. Metrics include rssi, throughput, pkt_size, xmit_rate, sender_ip to name a few. Example of filters would be average, sum, etc.

Additional hardware and software for testbeds include virtual machines and Linux 2.6.35 kernel nodes. The KVM virtual machines and the virtual machine aggregate manager are stored on the VM server. Lastly, to complete the wireless system for WiMAX enabled beds there is a NEC base station transceiver system, consisting of an indoor and outdoor unit, and base station servers. The servers store software that enable integration and functionality of the entire WiMAX system and manage multiple remote access, operation, and configuration of the network. This includes CLICK, a software defined router, the ASN-GW Controller, and the WiMAX RF aggregate manager that are stored on the base station server.

The Linux nodes are custom built system that include a 1 GHz VIA C3 processor that has 512 MB of RAM, a 20 GB hard disk, two 100BaseT Ethernet ports - one for control and one for experiment, two wireless mini-PCI 802.11a/b/g interfaces, and an integrated chassis manager that works with the CM for remote monitoring of hardware. WiMAX enabled nodes include Intel Centrino Advanced + Wireless 6250 USB devices. The user has root access to the Linux nodes and they can be configured with whatever modules and software an experimenter desires as long as it is supported by the hardware. The virtual machine image can be saved and stored on the repository for imaging of nodes in future.

The NEC base station is proprietary but user configurable. The BCR system parameters are just one of many parameters an experimenter can set to dictate BS behavior. Changing these parameters on the BS, changes the parameter on all SS. The configuration that is passed to the BS, it the same configuration that the BS will pass to every SS on the network via MAC management messages. Though this is useful for a WiMAX experimenter, it does not allow for any malicious activity to simulated. Therefore, we still needed to find a way to set the system parameters on attacker nodes separate from the BS settings of system parameter. An exhaustive list of the experimenter configurable BS parameters can be found at:

http://wimax.orbit-lab.org/wiki/WiMAX/17/00Wireless#WirelessControlServices

The parameters that are important to this study our listed in Table 2.4 with explanation of their importance. The remaining sections will detail experiments and investigations that have been conducted using the ORBIT resources.

Table 2.4: ORBIT base station parameters used

PARAMETER	DESCRIPTION	DEFAULT
reqbackoffstart	Initial backoff window size for contention bandwidth requests, expressed as a power of 2. This is encoded in the UCD message in the MAC layer. [0..15]	3
reqbackoffend	Final backoff window size for contention bandwidth requests, expressed as a power of 2. This is encoded in the UCD message in the MAC layer. [0..15]	15
dcducdinterv	Time interval between UCD and DCD mesages (units: frames) [10..4000]	200
odunoisefloor	Outdoor unit noise floor [0..100]	40

2.5 Summary

This chapter summarized the details of the IEEE 802.16e standard and WiMAX protocol. Coverage of the MAC layer, with specific focus on the UCD and UL-MAP messages, and TLV encoding was presented. The information in these sections communicate the importance of our study of WiMAX security and the protocol intricacies that will be used in finding and understanding the BCR system parameters. The security of the MAC layer was briefly explained to provide further context to the previous Section 1.3, past work in WiMAX security. Next, an exhaustive look at the BCR system parameters was given to help understand the hypothesis that system parameters have potential to be used as a DoS attacks and why modifying them in hardware has been a tedious job.

Additionally, the original work, that was the catalyst for this study, was presented and explained followed by an overview of the hardware, software, and facilities used for the hardware implementation. This information, as well as Section 2.1.2, serve as an introduction to the experiments discussed in the next chapter, Chapter 3, and will allow one to fully comprehend the methods we have used in our search for the BCR system parameters in hardware. These methods are presented in the remaining Chapters 4, 5, and 6.

Chapter 3

ORBIT Testbeds

As eluded to in Chapter 2 and Section 2.3, WiMAX hardware experiments were a natural progression after software simulations. There were multiple motivations for hardware experimentation. First, carrying out the same experiments on real hardware would provide a good comparison to software and hopefully validate the software results. Second, it would provide analysis of the reliability of a popular software network simulator, the NS-2. Additionally, all the reasons for investigating the BCR parameters in [7] were motivation for carrying out hardware experiments. Being able to replicate attacker behavior in hardware proved to be hard. While we applied various techniques to find a method that would allow us to mimic their actions, we analyzed the effect of BCR system parameters when no malicious activity is simulated.

This chapter presents these hardware experiments which were conducted on the ORBIT resources. We begin by introducing the experimental setup in the following section. Next, we look at the two testbed environments, indoor and outdoor, that were used for these in Section 3.2 and 3.3 respectively. To conclude this chapter, the actual analysis of the experimental data of these two environments is presented in Section 3.4 and the differences seen in results is explained.

3.1 Introduction

A few changes were required for hardware experimentation though. We no longer consider frame duration because it is fixed at 5 ms due to restrictions of WiMAX certified equipment. Secondly, the NEC BS does not provide *request retry* as a configurable parameter. *Backoff end* is used

instead because, as explained in Section 2.2, both *request retry* and *backoff end* determine when packets are dropped. Additionally, for these experiments no malicious activity was simulated. Instead all WiMAX nodes use the same BCR system parameters. The parameters settings used for these experiments can be seen in Table 3.1.

Table 3.1: Indoor vs outdoor experiment parameter settings

Parameter	**Treatment 1**	**Treatment 2**	**Treatment 3**
frame duration (s)	0.005	0.005	0.005
dos_backoff_start	1	3	5
dos_backoff_end	2	6	10

WiMAX enabled resources at Rutgers University's WINLAB facilities are being used to conduct hardware experiments. The resources are part of ORBIT [40], which is part of larger infrastructure of the GENI network. ORBIT has two NEC WiMAX BSs that we are using for our experiments. The SSs at ORBIT consist of laptops running Linux Kernel 2.6.35 with ntel® Centrino® Advanced + Wireless 6250 devices providing WiMAX service. Experiments consists of one BS, eight or more traffic generating SSs, and one sink node directly connected to the BS that acts as a receiver for all traffic generated by SSs. The traffic generating nodes use Constant Bit Rate (CBR) generators, to transmit 1024 B UDP packets every 0.5 s. The OTG and OTR were used as the transmitters and receiver. For each experiment, traffic is generated for 120s and OML [44] is used to collect measurements of the incoming traffic at the receiving node.

3.2 Indoor Testbed

ORBIT provides an indoor BS and an outdoor BS. The indoor BS and its corresponding WiMAX network have most environmental conditions controlled. The indoor BS, eight nodes that can be configured to be a WiMAX network and an Ethernet sink node, are enclosed in a Faraday Cage preventing outside interference from other wireless communication. Coaxial connections and tunable attenuators imitate the air medium between the BS and SSs. The attenuators can be used to simulate fading or varying signal strengths if desired, but no attenuation is used in our experiments. The indoor testbed, with all attenuation set to zero, produces a fairly equal receive signal strength, within +/- 2 dB, at each SS.

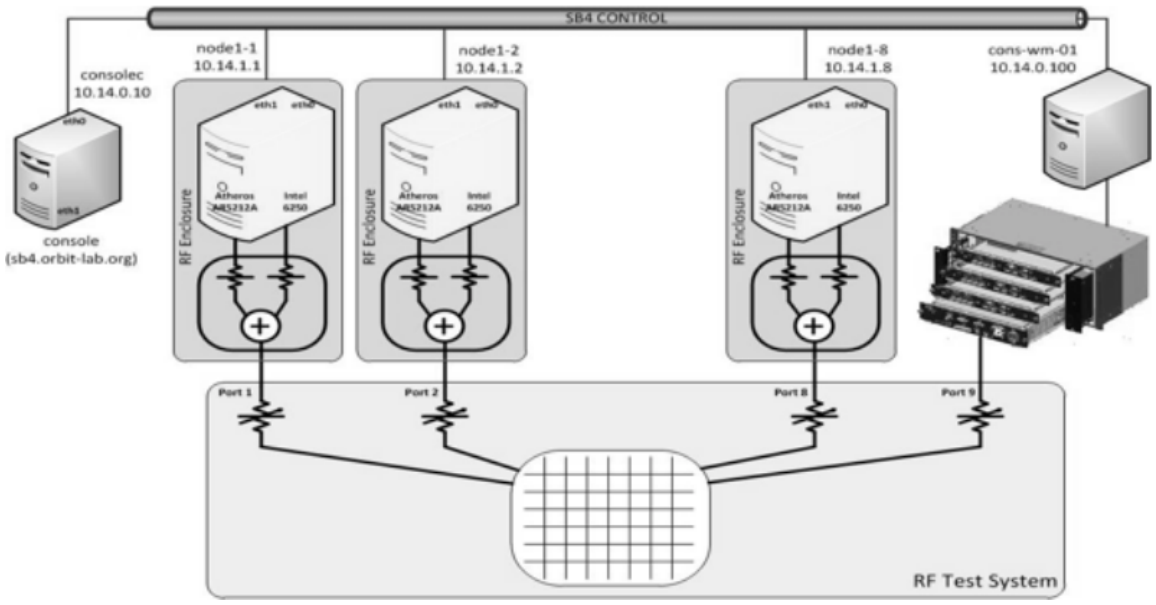

Figure 3.1: ORBIT's indoor WiMAX testbed, Sandbox 4 [45]

3.3 Outdoor Testbed

The outdoor BS and testbed consists of forty plus SSs deployed at various locations throughout Rutgers University's Busch Campus, Piscataway, New Jersey. The testbed includes both fix subscriber stations and mobile stations (MS). This testbed resembles a real WiMAX network with SSs at various locations and distances from the BS and experiencing a wide range of receive signal strength. Transmission paths are vulnerable to environmental factors and there is an increased amount of traffic compared to the indoor testbed. For consistency, we use the same eight WiMAX nodes and sink node in our experiments; the WiMAX nodes include node1-1 through node1-8 and sink node is node 4-1.

3.4 Indoor vs Outdoor

The following quantitative and qualitative analysis of the indoor and outdoor testbeds is based on the data that we have collected on both testbeds. The end of the previous section describes the parameters of this data. Data has been collected and analyzed for 26 experiments on each testbed. Table 3.4 presents the average throughput and packet-loss rate for all nodes involved in the experiments for the outdoor and indoor testbeds. Refer to Appendix C for detailed results of each

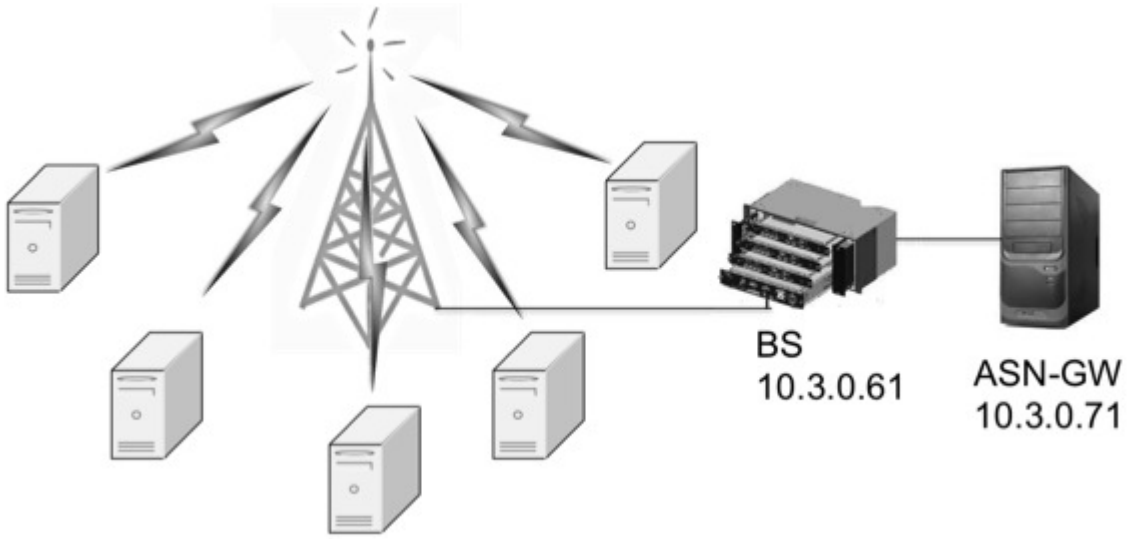

Figure 3.2: ORBIT's outdoor WiMAX testbed, Sandbox Outdoor

experiment run.

As expected, the data indicates that the indoor testbed achieves higher throughput for the receiving node and has a lower dropped packet rate. The indoor testbed produces a fairly consistent throughput for each SS with the average throughput for all eight nodes of 2031.2086 B/s and a range of 11.4872 B/s. The average throughput for outdoor was only 14.325 B/s lower than indoor, but the throughput range was much larger, 131.9 B/s.

Significant performance differences between nodes of the outdoor testbed are evident. For example, node 7 performed worst with an average throughput of 1919.895 B/s in comparison to node 3s average of 2051.572 B/s. Reasons for this include increased traffic activity from other parts of network and varying receive signal strength (RSSI) at each SS. The largest recorded range of RSSI for the outdoor nodes used in our experiments is 19 dB, -65 to -46 dBm. The results seen in the outdoor environment, large throughput range and varying node performance, can be accounted for by the environment factors that the outdoor testbed experiences that the indoor testbed does not. The nodes that performed worse likely are further away from the BS operating with a lower receive signal strength and/or experience such things as multipath fading.

Performance fluctuation can also be seen between groups of experimental runs. ORBITs network testbeds are still under development and are constantly undergoing modifications to produce better network performance and results. Implications of this will be discussed further in the following

Table 3.2: Average throughput and packet-loss

Testbed	Node	Avg. Throughput (B/s)	Avg. Packet Loss (B/s)
INDOOR	Node 1	2038.153823	15.42564103
	Node 2	2035.856393	17.39487179
	Node 3	2031.910245	21.00512821
	Node 4	2030.59743	18.70769231
	Node 5	2030.605125	18.05128205
	Node 6	2028.964095	19.69230769
	Node 7	2026.994868	21.66153846
	Node 8	2026.666663	21.98974359
	Receiver	16271.83728	161.4769231
OUTDOOR	Node 1	2020.079463	67.65714286
	Node 2	2034.856374	40.22857143
	Node 3	2051.571781	4.594871795
	Node 4	2048	2.438095238
	Node 5	2016.164103	59.12380952
	Node 6	2047.015385	1.219047619
	Node 7	1919.67179	237.1047619
	Node 8	1997.784612	91.42857143
	Receiver	16111.26115	504.6857143

section.

The ANOVA analysis presented in Table 3.4, 3.4, 3.5, and 3.6 highlight the significant differences that are expected between an outdoor, real WiMAX network, and indoor, an ideal network, environments. In these tables, X1 and X2 represent *bw_backoff_start* and *bw_backoff_end*, respectively, SS is the sum of squares, DF is degrees of freedom and MS is mean square. F0 is the mean square of the specific parameter divided by the mean square of error. In this test, the null hypothesis is different levels of each parameter or parameter combination affect the throughput or packet-loss rate equally, meaning that the value/s of the specific parameter or parameter combination does not significantly affect the measured quantity. A confidence of 95% is used for our analysis, α equals 0.05, and determines when to accept or reject the null hypothesis.

Table 3.3: ANOVA table for average throughput, indoor

SOURCE	**SS**	**DF**	**MS**	**F0**	$\mathbf{F_\alpha}$	**Prob>F0**	**R-SQUARE**
Backoff_start, X1	72.7	2	36.37	0.07	3.5915	0.9315	0.0046898
Backoff_end, X2	204.8	2	102.42	0.2	3.5915	0.82	0.013207
X1*X2	6536.4	4	1634.1	3.2	2.6547	0.0393	0.42145
Error	8673.5	17	510.2				
Total	15509.2	25					

Table 3.4: ANOVA table for average packet-loss rate, indoor

SOURCE	**SS**	**DF**	**MS**	**F0**	$\mathbf{F_\alpha}$	**Prob>F0**	**R-SQUARE**
X1	131.2	2	65.6	0.14	3.5915	0.8671	0.0098891
X2	160.8	2	80.38	0.18	3.5915	0.84	0.012116
X1*X2	5237.9	4	1309.47	2.87	2.6547	0.0551	0.39479
Error	7757.4	17	456.32				
Total	13267.6	25					

If F0 is more than F_α, than the null hypothesis is rejected and that parameter or combination of parameters does have significant affect on the quantity measured, this is known as the F-test. Similarly, if the probability that an F is greater than F0 (Prob > F0), also known as the *p-value*,

Table 3.5: ANOVA table for average throughput, outdoor

SOURCE	SS	DF	MS	F0	F_α	Prob>F0	R SQUARE
Backoff_start, X1	94.1	2	47.06	0.2	3.5915	0.9798	0.0020603
Backoff_end, X2	1798.8	2	899.4	0.39	3.5915	0.6824	0.039373
X1*X2	4872.6	4	1218.16	0.53	2.6547	0.7158	0.10666
Error	39115.1	17	2300.89				
Total	45685.9	25					

Table 3.6: ANOVA table for average packet-loss rate, outdoor

SOURCE	SS	DF	MS	F0	F_α	Prob>F0	R SQUARE
X1	99.4	2	49.69	0.02	3.5915	0.9785	0.0021924
X2	1785.8	2	892.91	0.39	3.5915	0.6824	0.039395
X1*X2	4792.2	4	1198.05	0.52	2.6547	0.7193	0.10571
Error	38842.8	17	2284.87				
Total	45331.5	25					

is less than α, the null hypothesis should be rejected as well. The R-Square value is defined as the percent of variance explained by a parameter or combination of parameters.

When comparing F0, F_α, and R-Square, one can conclude that *bw_backoff_start* and *bw_backoff_end* and their combination does not have any significant affect on throughput or packet-loss rate in the outdoor environment. Interestingly, the second-order effect of combining these parameters does have a significant affect in the indoor environment. Neither parameter alone will affect throughput or packet-loss rate, but the combination of the parameters can be significant in ideal WiMAX network but none are significant in real WiMAX networks. These results can be visualized in Figure 3.3. The R-SQUARE values for both indoor and outdoor are plotted in this figure. Despite this, it is crucial to note that according to ANOVA analysis the outdoor results are not significant.

These results also show the importance of real hardware network testbeds like ORBIT and GENI for system testing. According to our results, the environmental issues play a significant role in determining the affects of system parameters. When environmental conditions are modeled the significance of certain parameters change. For complete system testing, this is essential to know.

Figure 3.3: Piechart of indoor and outdoor test results

Many software simulators only allow replication of ideal networks and they are not able to mimic conditions of multipath fading and other environmental factors. With such software simulators, it is very critical to consider hardware experimentation as well to fully understand the effects.

3.5 Summary

This chapter presents in detail the WiMAX network environments of Rutgers University's ORBIT testbeds. These are the testbeds used for all of our hardware experiments and investigation into the BCR system parameters in hardware. Additionally, an analysis of the indoor and outdoor environments, testbeds SB4 and outdoor, was presented. The analysis did not simulate malicious activity. Instead it looked at the effect of different settings of two BCR system parameters, *backoff_start* and *backoff_end*. The individual parameter settings had no significant affect on the average throughput of the SSs, but the interaction of these two parameters did affect throughput in the indoor environment only. This magnifies an interesting difference between indoor and outdoor network testing.

The indoor testbed is controlled to produce almost ideal network conditions and this is similar to the ideal conditions that software network simulators create. The outdoor testbed is a

real WiMAX network. The varying results we see between these two environments exemplifies the importance of hardware and real network testing. Vulnerabilities present in one environment may not exist in another. It is always important to know how protocols will act under real scenarios. Understanding the effects on real networks, allow administrators to focus on the true vulnerabilities for their networks rather than theoretical flaws that may have no effects.

As mentioned earlier, this analysis did not simulate attackers and the end of goal of this study is to simulate both client and attacker nodes. The reason for not simulating the attackers was due to an ongoing investigation of how to do this on hardware. It requires being able to modify the BCR system parameters separately from the BS. The next three following chapters will present the methods we applied on the ORBIT resources in an attempt to find and manage the BCR system parameters. The methods have been separated into three categories: Linux, Intel, and Protocols. Each will be discussed in separate chapters.

Chapter 4

Intel Centrino Advanced + Wireless 6250

The hardware experiment investigation began with the ORBIT resources. The ORBIT BS allowed us to set the BCR system parameters to whatever values we desired. Though we could control the value of the parameters by modifying them on the BS, all SS's followed this one set of values. We learned that simulating attackers that use a different set of BCR system parameters than the BS set requires additional work. This work includes modify the parameters after the BS sets them. The ORBIT nodes discussed in Section 2.4 use Intel® Centrino® Advanced + Wireless 6250 USB devices for WiMAX services. We first analyzed the software provided with the device that enables operation of the WiMAX hardware. We were looking to determine if the BCR system parameters could be found and modified in the software.

The Intel devices include the following software on the nodes:

- Driver source code, wimax-1.5.1,
- Firmware, i2400m-fw-usb-1.5.sbcf, and
- Utility tools, wimax-tools-1.4.4.

The driver source code package has an open source license providing freedom to make modifications if needed to control the BCR system parameters in our case. The firmware is a binary file that is not provided under an open license. The utility tools provide simple functions for

checking the WiMAX services. It is the driver package that was investigated in our work. Each Intel module will be discussed in detail in the Section 4.2. Following this, the techniques used to search and modifying the module and source code in search of the BCR system parameters is presented in Section 4.3. The two techniques include manipulation of the Intel source code and information gathering from the debug configurable parameters for the Intel device. We hoped to find the BCR system parameters and/or verify the settings of these parameters through these two options. If we could not control the system parameter by these means, at least being able to verify them on the SS nodes would allow us to determine if other methods of modifying the parameters were successful. Before delving into the two techniques in Section 4.3, we will first provide an overview of the Intel device and modules in the following Introduction section and Section 4.2.

4.1 Introduction

The driver source code package creates 3 loadable modules when compiled and installed into a kernel. These modules can be divided into two parts, a WiMAX kernel stack and driver for Intel i2400m. The driver is further modularized into a bus generic and bus specific driver. The following WiMAX modules, *i2400m*, *i2400m-usb*, and *wimax* serve the following purposes respectively: bus generic driver, bus specific driver, and WiMAX kernel stack [28]. These modules are dependent on *iwlagn* module, a driver commonly installed in the Linux kernel for Wi-Fi applications.

One must make sure that drivers are loaded into the kernel before being able to use WiMAX. Most system avoid wasting kernel memory by not keeping drivers in core when not in use and support for automatic loading and unloading of modules [41]. A useful function for loading modules is **modprobe**, a user-space helper function. It finds and loads other modules that the current module is dependent on to function. For example running the following command will load not only *i2400m_usb*, but also *i2400m* and *wimax*:

modprobe i2400m_usb

As a side note, one could configure **/etc/moudles.conf** for a particular module to automatically set module options upon loading, executing commands before and/or after loading, assigning an alias name, etc. For example, the alias could be set to wmx and/or debug levels of *i2400m* could be set to high when loading modules. The following section will discuss debug level of each module further and the purpose of each driver. The last two sections in this chapter examine two techniques

used for finding the BCR system parameters and information related to them. Both techniques are dependent on the WiMAX drivers and source code.

4.2 Driver Modules

The module *i2400m* forms the driver core and this part of the driver can additionally be divided into two parts, OS-glue and hardware-glue [28]. The OS part interfaces with the Linux operating system. The hardware part interfaces with the device according to the bus-specific driver. This type of modularity allows easy adaption of hardware module for various operating systems.

There are multiple debug parameters for this module. The debug level is user configurable with 0 being the lowest and 8 being the highest level. We created an image for the ORBIT testbed that had all debug levels set high. While trying to understand how and where the system parameters are handled, it was useful to have as much information as possible. The higher the debug levels are the more information about WiMAX's operation is logged. The following debug parameters are included with the *i2400m* module:

- dl_tx, transmit,
- dl_rx, receive,
- dl_rfkill,
- dl_netdev, network device,
- dl_fw, firmware,
- dl_debugfs, file system,
- dl_driver, the actual driver, and
- dl_control, the MAC management control.

These can be found within the Linux file system at:

/sys/kernel/debug/wimax:wmx0/i2400m

They can be read using the cat command and set by executing:

echo [0-8] > /sys/kernel/debug/wimax:wmx0/i2400m/[*debug_filename*]

Debug information is logged to two separate log files. One is specifically for WiMAX, while the other is a kernel log and contains information about various kernel operations as they execute. The two log files with directory path included are:

- /var/log/wimax/system_log
- /var/kern.log

This module also stores transmit and receive statistics. There are six statistics kept for each receive and transmit. They include total packets, minimum packets in the received/transmit buffer, maximum packets in the received/transmit packet, total received/transmit buffers, accumulated receive/transmit buffer size, minimum receive/transmit size (bytes), and maximum receive/transmit size (bytes). These statistics can be viewed by reading the following file:

/sys/kernel/debug/wimax:wmx0/i2400m_[*rx/tx*]_stats

The *i2400m* module has two additional debug configurable parameters. These are *reset* and *trace_msg_from_user*. All can be set by writing to the file associated with the parameter name. The variable reset will perform a device reset. A warm, cold, and bus reset are possible by writing 0, 1, or 2 relatively. Parameter *trace_msg_from_user* creates a trace of messages originating in user space to the trace pipe that *i2400m* module creates.

Configurable operational parameters for this module can be found at:

/sys/module/i2400m/parameters

These parameters include *rx_reorder_disabled*, *passive_mode*, *power_save_disabled*, and *idle_mode_disabled*. These parameters are defined in the IEEE 802.16e standard. The option *rx_reorder_disabled* if set, would no longer allow the driver to reorder receive packets that are received out of order resulting in an increase of corrupted data. With passive mode enabled, the driver will no longer be responsible for device setup. Instead, setup must be done via user space and caution must be taken to properly setup device. The *power_save* mode allows a mobile SS to negotiate periods of absence from the BS to save power. At these periods, the SS is unable to communicate via the uplink or downlink channel. Lastly, *idle_mode* allows a mobile SS with no data to transmit to shutdown all active communications and only periodically become available for downlink traffic [35]. This state can be requested by either the SS or the BS. The default configuration disables all of the

parameters and this can be verified by making sure a '0' is wrote to each parameter file. To enable these, one must write a 1 to the file by issuing the following commands on the node:

echo 1 > /sys/module/i2400m/[*parameter_name*]

We have been interested with the last two operational parameters *power_save_disabled* and *idle_mode_disabled* for reason that will be explained in Section 7.3.

An important feature of the universal serial bus (USB) is that it is just a communication channel between a device and the host. It does not require standardized meaning or structure in the data to function [41]. The *i2400m-usb* module defines and handles this communication. It also has it own debug parameters that specifically look at the USB communication interface. These parameters included:

- dl_tx,
- dl_rx,
- dl_notify,
- dl_fw, and
- dl_usb.

The *i2400m-usb* debug parameters can be found at that the following location:

/sys/kernel/debug/wimax:wmx0/i2400m-usb

The WiMAX stack provides common WiMAX control for current and future devices. The kernels handling of network device is protocol independent [41] and therefore requires a module that is common. This is a generic layer that provides uniform API to control different WiMAX devices including a user space management stack. The stack works by embedding a struct wimax_dev in the machine's control structures and uses the generic netlink to send API calls to userspace [29]. The netlink commands are basic and include sending operation messages to and from kernel to user space, killing and reseting operations, reporting a status change, and requesting current state. This module, similar to *i2400* and *i2400m*, has various debug parameters. These included:

- wimax_dl_stack,

- wimax_dl_op_rfkill,
- wimax_dl_op_reset, and
- wimax_op_msg.

These can be found at the following location:

/sys/kernel/debug/wimax:wmx0

Both debug parameters for *i2400m-usb* and *wimax* can be read and set in the same fashion as indicated for *i2400m* parameters. The only difference is the path directory.

4.3 Methods

Two methods for investigating the BCR system used the Intel® Centrino® Advanced + Wireless 6250 software. The first technique looked at and modified the source code.The second relied on setting the debug levels and analyzing the log files to see if the system parameters could be found.

4.3.1 Wimax-1.5.1

The wimax-1.5.1 package is a hierarchy of files written in C-language. They include source and header files. These file comprise the necessary code for integrating the WiMAX hardware and firmware of the Intel device with the Linux machine. Some of the hierarchy along with important directories, for our work, is depicted in Figure 4.1. We searched these files looking for parameter variables that may include the BCR system parameters.

Understanding how the Linux kernel handles network devices is useful when attempting to reverse engineer the source code. First, initialization of net devices is completed by defining a structure known as the net_device sturct [35]. This struct contains numerous parameters some of which are visible and some that are not. Rmem_start, rmem_end, mem_start, mem_end, some of the parameters within the net_device, represent the beginning and end memory addresses of shared memory used by the device. If receive and transmit use different memory addresses, rmem is for receive and mem is for transmit. Other interesting parameters that could be significant for security

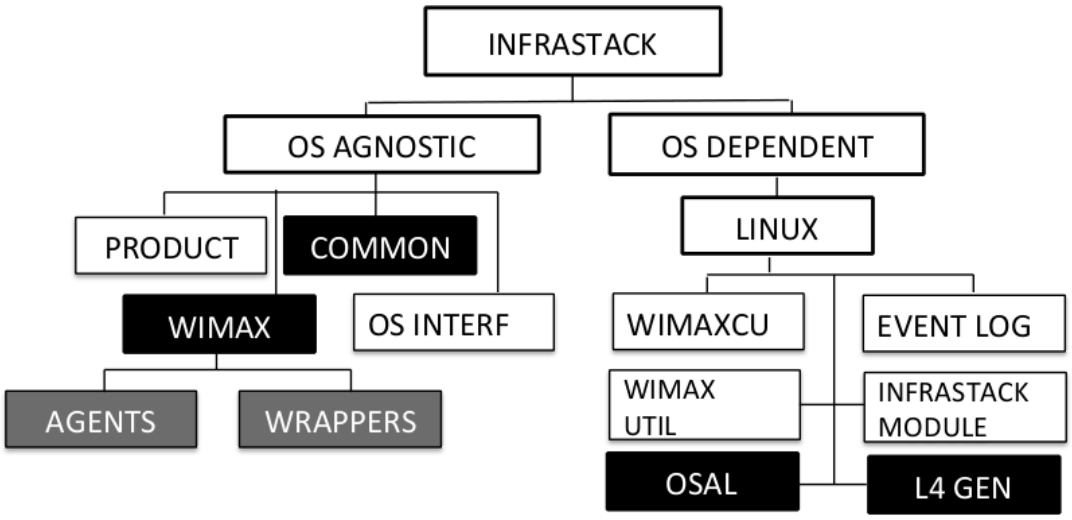

Figure 4.1: Directory hierarchy of Intel's wimax-1.5.1 source code

related work include *tx_queue_length* (max number of frames that can queued on the device transmit queue), *broadcast[]* and *dev_addr[]* hardware device address (MAC address).

There are also fundamental functions for operation of network devices such as *open*, *stop*, *set_mac_address*, *hard_start_xmit*, etc. The last function, *hard_start_xmit*, initiates transmission of a packet that is contained in a socket buffer [35]. Interaction between network drivers and the kernel occur one network packet at a time. A socket is a Unix abstraction to represent a network connection. Input/output buffers of any socket are a list of *sk_buff* structures. This buffer structure host network data throughput. Interrupts are used to handle receiving data. Other fundamental functions work in coordination with the function to build proper headers for transmitting packets.

When analyzing the source code, exhaustive searches were run on all files for keywords and parts of words such as *parameters*, *params*, *sk_buff*, *UCD*, *ULMAP*, etc. The source files containing these words are returned by the search. The variables and functions that were returned by these searches were further investigated in the context of the source code to see if they possibly could contain the BCR system parameters or information related to them. Some the parameters found that were promising and their associated files are listed in Table 4.1. The last column of this table indicates what these structures and/or functions define and handle according to the source code.

When there was a lack of or ambiguous information in the source code about suspected variables and/or functions, another method was used to gain more insight. Source files were modified to log parameters of interest using a function defined in the source code itself. The function used is

Table 4.1: Source code parameters investigated for BCR system parameters

PARAMETER	SOURCE FILE	INFORMATION CONTAINED
	Infrastack/OSDependent/Linux/L4Generated/DmGroupsHeaders.h,	
	Infrastack/OSDependent/Linux/L4Generated/L4BufMan_L3L4DmStructs_Desc.h,	
SF_AND_QOS_FLOW_MNGMT	Infrastack/OSDependent/Linux/L4Generated/L4BufMan_L3L4DmStructs.c	Harq and non-harq, UIUC (1-10 modulation and coding)
	Infrastack/OSDependent/Linux/L4Generated/L4BufMan_L3L4DmStructs_Desc.h,	
AggregatedBWRInfo,	Infrastack/OSDependent/Linux/L4Generated/L2DmMonitorGroups.h,	
IncrementalBWRInfo	Infrastack/OSDependent/Linux/L4Generated/L4BufMan_L3L4DmStructs.c	r2, numberOfBWRs
	Infrastack/OSAgnostic/WiMax/Agents/NDnS/L4_db/NDnSAgent_DB_Common.c,	
subscriptionParams_t	Infrastack/OSAgnostic/WiMax/Agents/NDnS/L4_db/NDnSAgent_DB_if.h	subscriber informatin and eap info
		home nsps, capl, rapl, name, server ID, channel plan, polling
nwParams_t	Infrastack/OSAgnostic/WiMAX/Agents/NDnS/L4_db/NDnSAgent_DB.c	interval, polling attempts, contention retries
nw_Params_t − > channel plan	Infrastack/OSAgnostic/WiMAX/Agents/NDnS/L4_db/NDnSAgent_DB_common.c	bw, duplex mode, ftt, channel size
	Infrastack/OSAgnostic/WiMax/Wrappers/NDnS/wmxSDK_Nds_Internals.h,	
wmx_ConnectParams	Infrastack/OSAgnostic/WiMax/Wrappers/NDnS/wmxSDK_Nds_3.c	nsp ID, connection type, home NSP, user credentials
		traffic priority, sf ID, max sustained traffic rate, max traffic
	Infrastack/OSDependent/Linux/L4Generated/DmGroupsHeaders.h,	burst, tolerated jitter, max latency, UG interval, min
qosParams	Infrastack/OSDependent/Linux/L4Generated/L4BufMan_L3L4DmStructs.c	resevered traffic rate, cid, sf scheduling type
	Infrastack/OSAgnostic/WiMax/Agents/NDnS/Source/NDnSAgent_L4P.c,	
	Infrastack/OSDependent/Linux/L4Generated/L4Common.h,	
ChannelInfoArray (channel plan)	Infrastack/OSDependent/Linux/L4Generated/L4BufMan_L3L4Structs.c	preamble, frequency, bw, channel id, fft
		pointer to OMA Paramerters: manufacture, model , fwVer,
pWimaxParams	Infrastack/OSAgnostic/WiMax/Agents/NDnS/Source/NDnSAgent_Monitor.c:	hwVer, mac
	Infrastack/OSDependent/Linux/L4Generated/DmGroupsHeaders.h,	
	Infrastack/OSDependent/Linux/L4Generated/L2DmMonitorGroups.h,	channel quality indicators: MIMOReportFlag, CQI
CQIParams	Infrastack/OSDependent/Linux/L4Generated/L4BufMan_L3L4DmStructs.c	measurements
MSChapParams	Infrastack/OSAgnostic/WiMax/Wrappers/NDnS/wmxSDK_Nds_3.c	nsp ID, password, username
	Infrastack/OSDependent/Linux/L4Generated/L3L4DmStructs.c,	
DcdUcdTimeout	Infrastack/OSDependent/Linux/L4Generated/L4BufMan_L3L4DmStructs_Desc.h	ms in frame
	Infrastack/OSDependanet/Linux/L4Generated/L1DmMonitorGroups.h,	
NormalULMap	Infrastack/OSDependent/Linux/L4Generated/L4BufMan_L3L4DmStructs_Desc.h	CRC Error, HCS errors
IdleParameter*,		
PreInitConfiguration,		
PLIParameters*,	Infrastack/OSAgnostic/Product/AppSrvInfra/DeviceConfiguration.c	Arq, harq, packet formatting information
		Open mobile alliance device management parametrs: radio
	Infrastack/OSDependent/Linux/L4Generated/L4BufMan_L3L4DmStructs_Desc.h,	module, terminal equipment, mac address, manufacture, hw
OMA_WimaxParams	Infrastack/OSDependent/Linux/L4Generated/L4BufMan_L3L4DmStructs.c	version, operator name, fw version,
linkLossParams	Infrastack/OSAgnostic/WiMax/Agents/NDnS/L4_db/NDnSAgent_DB_Save.c	stored nsp and network parameters for reconnecting

osallog(char ch, int flush).* The definition of this function is:

```
void osallog(char *ch, int flush)
{
static FILE *log = 0;

if (log == 0) {
log = fopen(OSALTRACE_FILE, "at");
if (!log)
             log = fopen(OSALTRACE_FILE, "wt");
if (!log) {
syslog(LOG_ERR, "wimaxd[osal] - can not open logfile (%s) for writing.\n",
OSALTRACE_FILE);
// release the lock if get any errors
pthread_mutex_unlock(&g_mutex);
return; // bail out if we can't log
}
 }
// write into file
fprintf(log, ch);
// put extra to log next line to build
//fprintf(log,"\n");
if (flush == 1)
fflush(log);

#ifdef OSAL_CONSOLE
printf(ch);
#endif
// release the lock once done with log
pthread_mutex_unlock(&g_mutex);

// fclose(log);
     }
```

This function was called where the parameters were accessed in the source code. For exam-

ple, the following code:

```
 char tmpBuffer[100];


sprintf(tmpBuffer, "nwParams_t Address: %d\nnwParams polling interval=%d, polling
attempt=%d\nsubscriptionParams Address: %d", nwParams_t, (*nwParams_t).pollingInterval,
(*nwParams_t).pollingAttempt,  subscriptionParams);
osallog(tmpBuffer, 1);
```

was added to the file:

Infrastack/OSAgnostic/WiMAX/Agents/NDnS/L4_db/NDnSAgent_DB.c

after *nwParams* was defined. After modifiying the source code, the software package had to be reinstalled on the Linux node for the change to take effect. To reinstall the WiMAX software the following commands must be entered in the order listed below:

1. cd wimax-1.5.1
2. ./configure –prefix=/usr –with-linux=/usr –with-libwimaxll=/usr –sysconfdir=/etc -localstatedir=/var –enable-instrument
3. make
4. make install

After the source code was modified, the value of *polling attempt* and *polling interval*, both defined in *nwParams*, and the memory addresses of *nwParams* and *subscriptionParams* are printed to the log file when called in source code. The logging of these parameters can be seen in Figure 4.2. Varying the BCR system parameters and comparing the log files for different parameters settings, indicated no change in the *nwParams* variable. Since there was a change in the BCR system parameters but no change in the variable under investigation, we were able to determine that the system parameters were not stored within the variable. This method of comparison was used repetitively for different methods and is described in more detail in the following Subsection 4.3.2.

Figure 4.2: Log file from modified source code

4.3.2 Debug

Figure 4.3 shows the process that was used to wean information about the system parameters from the log information. First, the system parameters, *backoff_start* and *backoff_end*, had to be set. The default parameters were always used to begin, *backoff_start* set to 3 and *backoff_end* set to 15. Next, nodes were configured and, specifically, the debug levels were verified and set on the nodes. The debug image that was created could be used and loaded onto the nodes. But when interested in what information a specific debug level would give, one needs to set only this debug level. When nodes and debug levels are configured, the nodes are connected to the WiMAX network, traffic is passed and received, and then the nodes are disconnected from the network. After this the system parameters are modified one at a time and the process is restarted. At least two nodes are used so that traffic can be passed. Stepping through this process allows the log files to log information during all WiMAX processes that would be executed during an experiment: connection, passing traffic, and disconnection.

For each cycle a separate log files is saved so that comparisons can be made. To keep log files from becoming full of other information while setting up for the next run and to separate between each cycle. The tail command issued on the log files was used to write logged information to a separate file. An example of this command is:

$$\textbf{tail f } [\textit{log\_filename}] >> [\textit{new\_filename}]$$

At least two runs and the corresponding log files, each with different system parameter values, were used to do comparisons. The log files contain a variety of information including the data packets sent to device from user space and received from the BS. We were looking for packets in the log that had bytes change along with the change in the system parameters. In this way, we

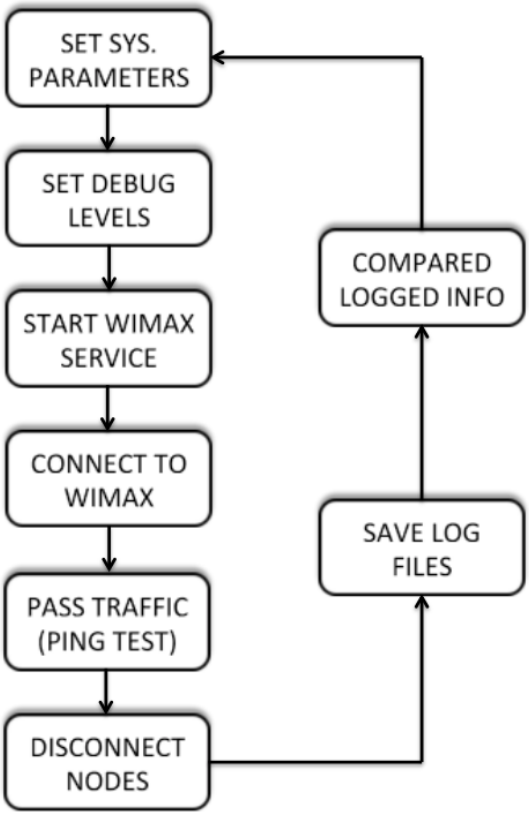

Figure 4.3: Flow chart of process used to find BCR system parameters

were attempting to find the UCD message and the system parameters setting set by the BS in this MAC management message. Specifically, data packets of the same size were analyzed to see if at least one or two bytes vary indicating the system parameter change. The network configuration used in our experimentation was always the same allowing us to assume that the UCD message size at the various stages of connection would be constant throughout experiments.

Changing the system parameters in a controlled fashion aids in the ability to use this technique for finding them. For example, first changing only one system parameter, *backoff_start*, by a value of one and keeping *backoff_end* the same allows us to look for one small change in the logged data. Typically, we would then set *backoff_start* back to default and decrease *backoff_end*. This method was used throughout our entire investigation for the BCR system parameters.

There was an overwhelming amount of information logged when the following debug levels were set to a maximum value for all modules:

- dl_rx,

- dl_tx,
- dl_control,
- dl_driver,
- dl_usb, and
- dl_fw.

With use of the **i2400m.h** header file located in the following directory **/include/linux/wimax**, we were able to isolate logged information from the time the first successful connection command was sent until the disconnect command was sent. Within this time interval, investigation of packets labeled *rx_cntl* were compared and two potential packets were found that may carry the BCR system parameters. Further investigation is required to confidently determine that these logged messages store the system parameters.

Setting a minimum number of debug parameters will make future analysis easier. We determined that the important debug values for potentially gaining information about the system parameters *backoff_start* and *backoff_end* include dl_rx and dl_control. The other debug parameters set for our logs provided no insight to the parameters. Additionally, the use of both dl_rx and dl_control provided redundant information about the the received packets.

4.4 Summary

Intel® Centrino® Advanced + Wireless 6250 USB device is reviewed in this Chapter. Presented in two sections are the techniques we used to gain information about how the WiMAX device and Linux drivers provided for the device handle the BCR system parameters. The debug parameters, Section4.3.2, and the wimax-1.5.1 source code, Section 4.3.1 provide abundant information about how this WiMAX device works. Though we have reason to believe the system parameters can be verified via the log files. Soon we will see in Chapter 6, are study suggests that the BCR system parameters of interest are managed by the firmware only. It appears they are only passed to the SS node as information packets for debug logs and not MAC control messages. This result continued our search for a means to modify the BCR system parameters which leads us to Chapter 5.

Chapter 5

Linux

After searching the source of the Intel® Centrino® Advanced + Wireless 6250 USB devices and not finding a variable or function that directly handled the BCR system parameters, we next looked to the Linux systems running on the SS nodes. The Linux system interacts and manages the peripheral devices that are installed on the SS nodes. For this reasons, we hypothesized that the BCR system parameters may be passed by the WiMAX device to the Linux operating system and stored there. Though are search of the source code did not uncover the system parameters, it was plausible that a communication function had been overlooked that passed the parameters to the Linux system. With this in mind, we dug into the Linux file system and memory in hopes of discovering the BCR system parameters.

The ORBIT nodes run Linux kernel 2.6.35. Linux is a free, open-source operating system that is written and built in C programming language. Open-source software gives users the luxury of seeing how the system is implemented and the freedom to change the code if needed or desired. When granted root-access to a Linux kernel, such as researchers are on ORBIT nodes, one can change and modify the operating system in any way. The open-source platform and root privileges allowed flexibility in the techniques we applied to investigate the BCR system parameters on Linux. This freedom was needed because the WiMAX protocol does not allow you to override the BS mandated system parameters. If the BCR system parameters were found, unauthorized modifications would have to be made to them. Modification would allow attackers to have a different setting than the client SS's that continue to appropriately follow the BS's settings.

As seen in Chapter 4, we were able to modify the open-source drivers developed for Linux

systems. In this Section we look at two more methods used to locate the BCR system parameters and understand how they are handled and accessed. The first, leverages the information stored within the Linux file system, Section 5.1. Special emphasis was given to the **/proc** directory and is explained in Subsection 5.1.1. The second method, uses GDB, a common Linux debugging tool, to hunt for the parameters within the memory of the Linux node, Section 5.2.

5.1 File System

On a UNIX or Linux system, everything is a file; if something is not a file, it is a process [15]. This statement is true but can be misleading if ones definition of a file does not include unordinary files such as directories, special files, links, sockets, and named pipes. After drivers, file systems are the most important class of modules in a Linux system [41]. The typical Linux files system structure can be seen in Figure 5.1.

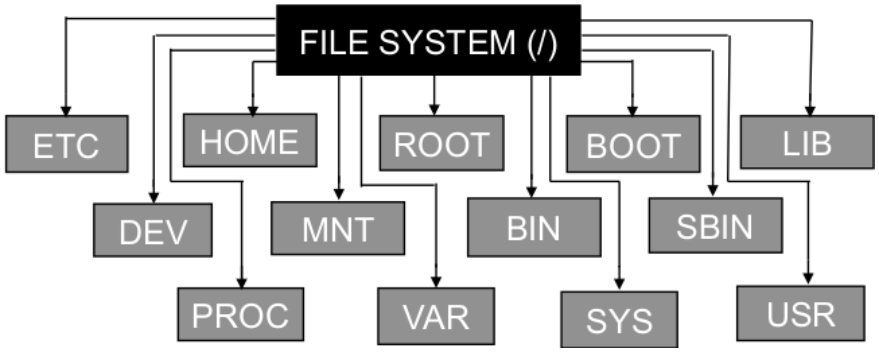

Figure 5.1: Linux file system hierarchy

The **/proc** file system is created by software and used by the kernel to export information to the world. Each file relates to a kernel function that generates file contents in **/proc**. Many command line functions simple read **/proc** files. For example, the command *lsmod* gets the information from **/proc/modules** and displays it in a user-friendly format. Device drivers export information via **/proc** [41]. Clearly the file system is read from, but it can also be written too. Most **/proc** files are intended to be read-only but forced memory overwriting can provide a means around this. If the BCR system parameters were found in the **/proc** directory, we could use this technique to overwrite them and set them to whatever parameter settings we choose.

Drivers and modules get classified into classes in Linux. Three common classes include char

devices (accessed as stream of bytes similar to file), block devices (sends data in randomly accessible fixed size data blocks similar to a disk), and network interfaces. The last type encompasses devices that can exchange data with other hosts [41]. Unfortunately, network interfaces do not easily map to file system such as char and block devices. Instead of having input and output assigned to a file within the **/dev** directory, they are assigned unique names. The name for WiMAX device is *wmx*. Information gets stored as files under the interface name.

Within the Linux file system is various WiMAX related files. Some are configuration files, other log files, but we are attempt to find out if the system parameters of BCR are stored somewhere within the file system. The Linux file system is displayed as a tree in Figure 5.2. The promising parent directories include:

- /proc - interface to kernel data structures,
- /var - system writes to files within during operation,
- /sys - exports kernel device information to user space,
- /usr - secondary hierarchy for user data, and
- /etc - configuration files for Linux system.

Each of these directories stores information related to WiMAX and network devices. Chapter 4 looked at the log files contained within the **/var** directory and the WiMAX configurable operational parameters that are stored in **/sys** directory. Also in the **/var** directory are WiMAX database and definitions files and **/etc** directory contains a WiMAX configuration file. The file name and path directories are:

- /var/lib/wimax/WiMAX_DB.bin,
- /var/lib/wimax/ WiMAX_Def.bin, and
- /etc/wimax/config.xml.

The *backoff start* and *backoff end* system parameters were not found in any of these WiMAX files. The **/usr** directory contains WiMAX security information such as keys and certificates, but again, does not contain the system parameters leaving only the **/proc** directory, which the next section will look at in great detail.

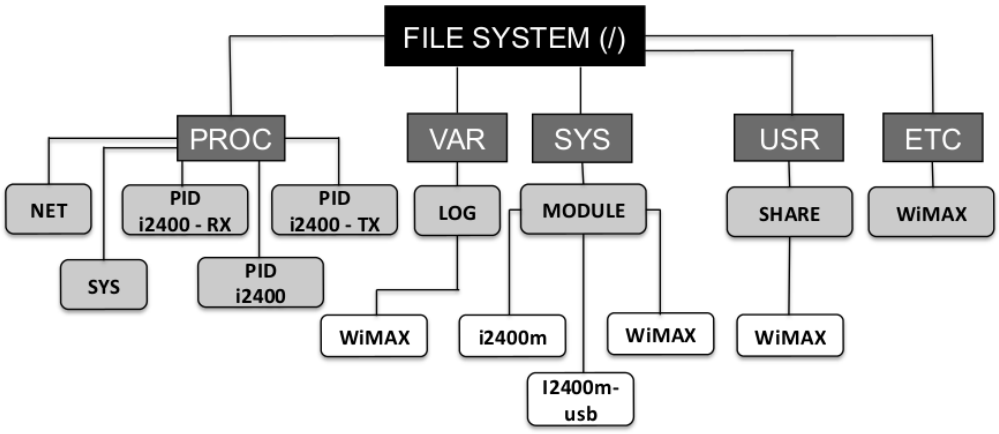

Figure 5.2: Linux directory path of WiMAX related files

5.1.1 Proc

As mentioned earlier, the **/proc** directory contains an abundant amount of information about the Linux operating system. It contains information about registered resources with **/proc/ioports** and **/proc/iomem** [41]. Memory assignments are saved within the **/proc** directory, and each running process has its own directory there. Another useful and important part of information that is stored here is the kernel symbol table.

At load time, any unresolved symbols used in a module are linked to the kernel symbol table. The addresses of global kernel items such as functions and variables are stored in this table. This allows drivers to be implemented in a modularized fashion. For example, the three Intel WiMAX drivers are modularized and stacked on top of each other. The *i2400m* is dependent on *i2400m-usb* and *wimax*. We see the symbols again in the memory dumps that look for the BCR system parameters in the next section.

Scripts were used to write the contents of files from this directory to a text file that was later compared for different system parameter configuration. We were looking for a specific file that changed to reflect system parameter changes. The thought was that controlling the system parameters could be accomplished by overwriting this specific file. Entries within the **/proc** directory included:

- /proc/sys/net/ipv4/conf/wmx0/[*filename*]
- /proc/sys/net/ipv4/[*filename*]

- proc/*i2400m_usb**
- proc/*i2400m-rx**
- proc/*i2400m-tx**
- proc/*iwalgn**

The files with the '*' at the end are process identification numbers (PIDs). These numbers change each time that the WiMAX service is started. The bash script uses **grep** to search the output of the command **ps aux**, which returns process information, to find the iwlagn and i2400m processes. The text output from the script can be seen in Figure 5.3. There was no entry found within **/proc** that corresponded to the system parameters *backoff start* and *backoffend*. The three WiMAX related process were always found to be asleep and not active. It is possible that more information is stored in these entries when the processes are active. A different script or code would need to be developed to log the file entries within the process directories when the process becomes active. Our script only logged the information when the the process were asleep.

```
30000
/proc/sys/net/ipv4/neigh/wmx0/delay_first_probe
5
/proc/sys/net/ipv4/neigh/wmx0/gc_stale_time:
60
/proc/sys/net/ipv4/neigh/wmx0/locktime:
100
/proc/sys/net/ipv4/neigh/wmx0/mcast_solicit:
3
/proc/sys/net/ipv4/neigh/wmx0/proxy_delay:
80
/proc/sys/net/ipv4/neigh/wmx0/proxy_qlen:
64
/proc/sys/net/ipv4/neigh/wmx0/retrans_time:
100
/proc/sys/net/ipv4/neigh/wmx0/retrans_time_ms:
1000
/proc/sys/net/ipv4/neigh/wmx0/ucast_solicit:
3
/proc/sys/net/ipv4/neigh/wmx0/unres_qlen:
3
```

```
Idx Device                Multicast Address                 Source Address    INC    EXC
proc/622/net/netlink:
sk       Eth Pid    Groups   Rmem     Wmem     Dump     Locks     Drops     Inode
f71a8200 0   0      00000000 0        0        (null) 2         0         6
f6aad400 4   0      00000000 0        0        (null) 2         0         5030
f69b5000 7   0      00000000 0        0        (null) 2         0         4504
f737ba00 9   0      00000000 0        0        (null) 2         0         4467
f737ae00 10  0      00000000 0        0        (null) 2         0         4447
f71ab800 11  0      00000000 0        0        (null) 2         0         153
f65c5600 15  366    00000002 0        0        (null) 2         0         6235
f71a8800 15  0      00000000 0        0        (null) 2         0         9
f6b00c00 15  376    00000000 0        0        (null) 2         0         7688
f6b00a00 15  373    00000001 0        0        (null) 2         0         6250
f71abc00 16  0      00000000 0        0        (null) 2         0         157
f490c400 16  8389585 00000000 0         0        (null) 2         0         7656
f490f600 16  12583889 00000020 65812    0         (null) 2         194       7657
f490d600 16  977    00000000 0        0        (null) 2         0         7653
f490fa00 16  4195281 00000020 0         0        (null) 2         0         7654
f71aba00 18  0      00000000 0        0        (null) 2         0         154
proc/622/net/netstat:
TcpExt: SyncookiesSent SyncookiesRecv SyncookiesFailed EmbryonicRsts PruneCalled RcvPruned OfoPruned
OutOfWindowIcmps LockDroppedIcmps ArpFilter TW TWRecycled TWKilled PAWSPassive PAWSActive PAWSEstab DelayedACKs
DelayedACKLocked DelayedACKLost ListenOverflows ListenDrops TCPPrequeued TCPDirectCopyFromBacklog
```

Figure 5.3: Text output from script that logged **/proc** directory information

5.2 Memory

The memory of Linux operating systems can be separated into virtual and physical memory. The virtual memory system allows for more addressing than physical memory would allow. Virtual memory is the address space and physical memory is actually present on the machine. Kernel and user space work with virtual addresses that are mapped to physical memory [2]. Page tables define

the exact memory mapping. Additionally, every peripheral device is controlled by writing and reading to and from the devices registers. A device typically has several registers and they are accessed at consecutive addresses in the memory or the I/O address space [41]. The memory address space of the Intel WiMAX modules can be determined in a couple of ways. The command **lsmod** gives the size of the modules and **/sys/module/[*module_name*]/sections** stores the memory offset. Both pieces of information can be found by reading the contents of **/proc/modules**, Figure 5.4. Given the total size and the memory offset it possible to dump the memory of the address space with proper software tools.

```
root@node1-2:~# cat /proc/modules
i2400m_usb 35737 0 - Live 0xf85e9000
i2400m 101253 1 i2400m_usb, Live 0xf8561000
wimax 33525 1 i2400m, Live 0xf8534000
arc4 12473 2 - Live 0xf8381000
iwlwifi 362374 0 - Live 0xf8a89000
mac80211 436455 1 iwlwifi, Live 0xf8a01000
i915 419126 1 - Live 0xf8581000
cfg80211 178679 2 iwlwifi,mac80211, Live 0xf8493000
drm_kms_helper 45466 1 i915, Live 0xf8486000
drm 197599 2 i915,drm_kms_helper, Live 0xf84c6000
i2c_algo_bit 13199 1 i915, Live 0xf8177000
snd_hda_codec_realtek 174313 1 - Live 0xf845a000
video 19068 1 i915, Live 0xf811c000
mac_hid 13077 0 - Live 0xf8045000
snd_hda_intel 32765 0 - Live 0xf8101000
snd_hda_codec 109562 2 snd_hda_codec_realtek,snd_hda_intel, Live 0xf8405000
snd_hwdep 13276 1 snd_hda_codec, Live 0xf8070000
snd_pcm 80845 2 snd_hda_intel,snd_hda_codec, Live 0xf839d000
snd_timer 28931 1 snd_pcm, Live 0xf810b000
snd 62064 6 snd_hda_codec_realtek,snd_hda_intel,snd_hda_codec,snd_hwdep,snd_pcm,snd_timer, Live 0xf8161000
soundcore 14635 1 snd, Live 0xf807a000
snd_page_alloc 14115 2 snd_hda_intel,snd_pcm, Live 0xf8075000
psmouse 96684 0 - Live 0xf8123000
serio_raw 13027 0 - Live 0xf8037000
lp 17455 0 - Live 0xf8024000
parport 40930 1 lp, Live 0xf802c000
e1000e 139997 0 - Live 0xf804c000
```

Figure 5.4: Contents of **/proc/modules** with WiMAX module information

GDB, the GNU project debugger, is a powerful interactive debugger used typically to trace a programs execution and execute one line at a time [15]. Beyond simple execution, GDB can be used to accomplish more complex tasks, such as reversing the execution, examining stack memory and symbols, and accessing memory of executing program. A feature of interest for our work is the ability to load the running Kernel in GDB. This allows one to examine the virtual memory used by the WiMAX drivers. The debugger is invoked as though the kernel were an application. All that must be provided to GDB to accomplish this, is the file name of the uncompressed kernel image and the core file [41]. An example of how to start GDB with the kernel as an application is:

gdb /usr/src/linux/vmlinux /proc/kcore

The **/proc/kcore** file represents the kernel executable in the format of a core file; it is a huge file that represents all kernel address space and all physical memory. GDB optimizes access by caching the core file information. GDB should be restarted to keep the core file cache up-to-date or it must be flushed. We used GDB to look at and write the memory of the Intel 6250 WiMAX drivers to binary files. Each time we ran an experiment and dumped the memory looking for the BCR system parameters, we restarted GDB. In this way the cache was update with each experiment run.

Linux loads the Intel 6250 drivers as three separate modules: *wimax*, *i2400m*, and*i2400m-usb*. Each time the system is restarted and/or modules are installed, they are loaded into a different part of memory. Though the start and end addresses differ, the memory structure and layout for each module remains similar, Figure 5.5. Each module is loaded into memory based on dependence, for example *i2400m_usb* is dependent on *i2400m*, therefore *i2400m* is loaded first into memory and assigned a lower memory address.

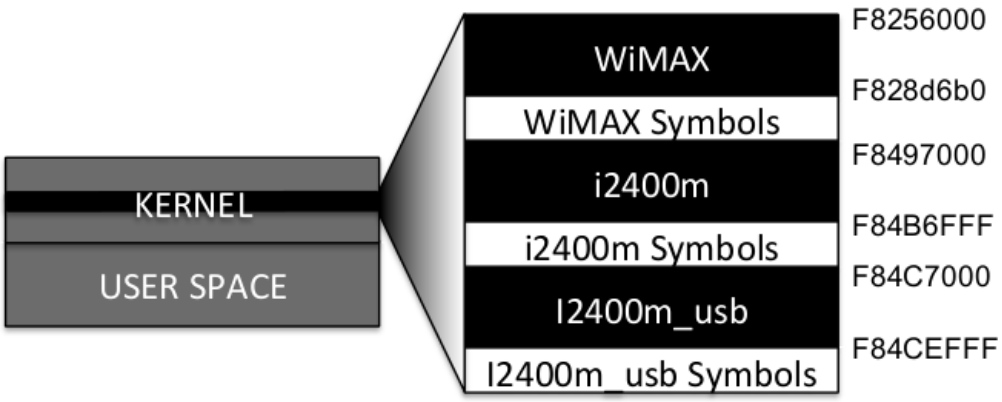

Figure 5.5: Example of memory stack of WiMAX modules

Memory dumps were made while the nodes were connected to the WiMAX network and passing traffic. The file comparison command **cmp**, which allows for binary comparison, was used to determine if the memory dumps for differnet BCR system parameter configuration showed any differences. Again, the files compared were the same indicating that the system parameters were not stored with the memory assigned to the WiMAX modules.

5.3 Summary

The Linux file system and memory was explained in this chapter. The various WiMAX files in the Linux file system that were dissected in search of the BCR system parameters are discussed. Also conveyed are the details of how we searched both the **/proc** directory and the Linux kernel memory for the BCR system parameters, *backoff_start* and *backoff_end*. Besides the log files discussed in Chapter 4, no information about the system parameters was found within the Linux file system or memory. This conclusion provides us no means to control the system parameters in hardware and again suggests that the system parameters may never leave the firmware on the WiMAX device. These results are not entirely unexpected because the IEEE 802.16e standard requires all SS to follow the BS's settings of the system parameters. Since one is not suppose to deviate from these settings, there is no direct, easy way to set the system parameters independently on the nodes and finding a way to do so in hardware is onerous task. Next, Chapter 6 will describe one last technique that we have exercised in hopes of finding a means to modify the system parameters.

Chapter 6

Reversing Protocols

This chapter presents one additional technique we used for collecting information about the BCR system parameters; traffic captures. Early on this method was applied to the WiMAX interface on the Linux nodes in ORBIT to hopefully capture the control and management messages, including the two MAC management messages of interest, the UCD and UL-MAP messages. There was some surprise to the lack of traffic captured on this interface, which led to further investigation of the Intel device, Chapter 4, and the Linux network stack, Chapter 5. The traffic seen suggest that only the firmware handles the BCR system parameters.

The captures at this interface are further discussed in Section 6.2. After fully considering the methods presented in the previous Chapters 4 and 5, we began to consider the possibility of man-in-the-middle attacks to manipulate the BCR system parameters. Following conversations with the ORBIT administrators, we determined that there was a wired interface between the BS and ASN-GW that would allow for man-in-the-middle attacks with support from the ORBIT administrators. The traffic captures and analysis of this interface is presented in Section 6.3. Before taking a more in-depth look at these traffic captures, Section 6.1 will provide more details of the resources use for traffic captures and the motivation.

6.1 Introduction

Network traffic captures can provide a myriad of information about how network protocols work including management and security, as well as end-user information, such as content being

communicated and devices. This technique of sniffing traffic is used by many types: malicious users to steal and monitor user information, developers to debug communications, and educators to teach people about communication protocols. There is many tools to complete traffic captures; tcpdump, Wireshark, and Tshark are just a few. All use the pcap application programming interface to manage traffic captures which in Linux systems is known as the libpcap library. Tshark [12] is a command-line interface tool that is offered in addition to Wireshark [13]. Due to the command-line interface of the ORBIT nodes, Tshark is used to complete network traffic captures.

Our desire to complete network traffic captures on the ORBIT testbeds was first initiated by a wanting to better understand how WiMAX management messages are handled on Linux nodes. It later turned into an investigation of whether or not man-in-the-middle attacks could be used to control the BCR system parameters. For these reasons traffic captures were completed on two different interfaces within the WiMAX network. The following two sections detail the captures of these two interfaces and the information that was gleaned.

6.2 Ethernet

WiMAX traffic was captured at the **wmx0** interface on the Linux nodes. The **wmx0** interface is the network device interface assigned to the WiMAX hardware. After turning on the WiMAX radios on the nodes and configuring the IP addresses for these interfaces, the traffic captures began. Traffic was collected while the nodes connected to the WiMAX BS, each node pinged all other nodes on the network and was pinged by other nodes, and then when it disconnected from the BS. These steps are depicted in the flow chart in Figure 6.1. During all of this of time, the only packets that were collected at the **wmx0** interface were the ping requests and replies on the network. On the outdoor testbed there was also miscellaneous broadcast messages and ARP requests from other devices on the network.

Figure 6.2 shows the details of a Internet Control Message Protocol (ICMP) ping reply that was carried over the WiMAX network. All WiMAX protocol headers have been stripped before receiving the packet at the **wmx0** interface. For this reason we do not see the BS identification number (BSID) anywhere in the packet, 44:51:DB:00:00:00. Four protocols are encapsulated at this interface: Ethernet, IP, ICMP, and the payload. The MAC addresses for the source and destination in the Ethernet header are identical and belong to the WiMAX interface, MAC address of the Intel

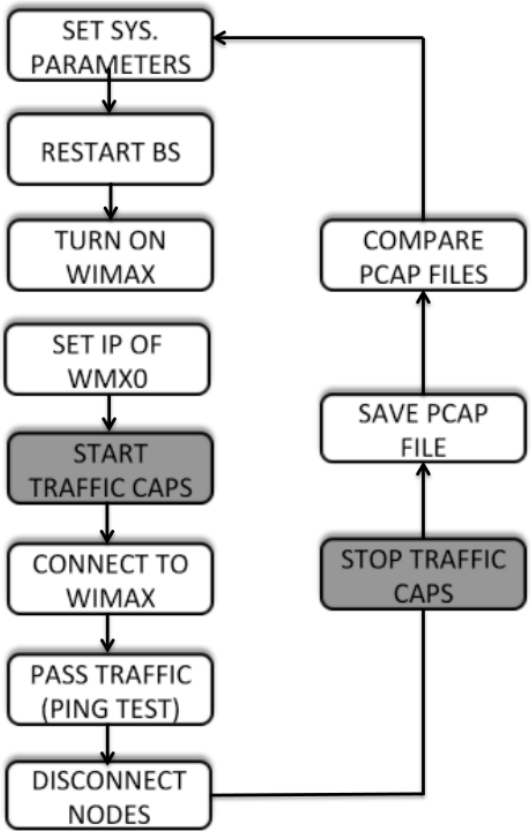

Figure 6.1: Flow chart of traffic capturing process on the **wmx0** interface

6250 WiMAX device. After the Ethernet header, is the IP header which contains the IP address of node 1 and node 2, 10.41.14.1 and 10.41.14.2 respectively. The ICMP and payload information follows the IP header.

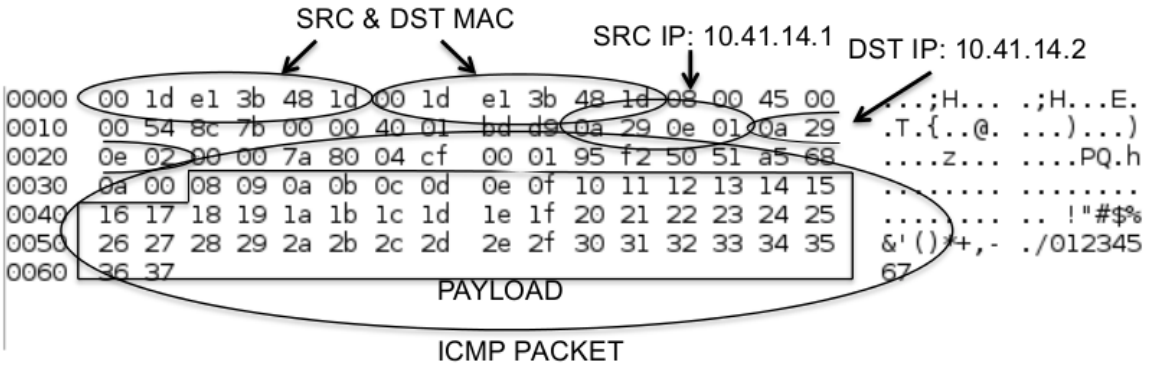

Figure 6.2: Ping reply captured on the **wmx0** interface

The limited packets seen at the **wmx0** interface indicates that none of the WiMAX MAC management messages are transferred to the Linux network device structure. Information about these messages may be encapsulated and sent via other communication to the kernel log. Due to the lack of control and management messages available on the nodes, we decided to look for another interface point. A point that was wired allowing for the possibility of a man-in-the-middle attack.

6.3 R6+ Protocol

One Ethernet connection in the ORBIT WiMAX network is between the Access Service Network Gateway (ASN-GW) and the BS. This provides one wired connection within the wireless network that could be used for a possible man-in-the-middle attack. The Access Service Network (ASN) and the ASN-GW handle and manage a plethora of important WiMAX management functions. The ASN is defined as a complete set of network functions needed to provide radio access to a WiMAX subscriber [9]. The ASN can include multiple network elements, one or more BS and ASN-GW [6]. Some of the required functions of the ASN are WiMAX Layer-2 (L2) connectivity with WiMAX mobile stations, transfer of authentication, authorization, and accounting (AAA) messages, authorization and session accounting for subscriber sessions, network discovery and selection of the WiMAX subscribers preferred network service provider (NSP), relay functionality for establishing Layer-3 (L3) connectivity with a WiMAX mobile station including IP address allocation, and Radio Resource Management (RRM). The ASN-GW is the logical entity that performs the control-plane functions including the RRM, radio resource and Quality of Service (QoS) management at user authentication.

Radio Resource Management refers to measurement, exchange, and control of radio resource-related indicators in a wireless network. This refers to the techniques and communication used to measure, estimate, and set radio resources. The RRM would be used to implement proprietary algorithms for radio resource allocation [9]. RRM is not always required because BS can implement many of the services initially, though it is required for handovers to support mobile stations. RRM messages are sent over what are referred to as reference points. Each reference point communicates between two varying end points. The reference point 6, R6, consists of the set of control and Bearer Plane protocols for communication between the BS and the ASN-GW. The Control Plane includes protocols for data path establishment, modification, and release control in accordance with the

mobile station events. The NEC BS uses a proprietary R6 protocol referred to as R6+.

An R6 protocol message of interest is the *RRM Spare_Capacity_Report*. This message includes the *RRM BS Info* which contains the *Full UCD Setting* [10]. Table 2.1 and 2.2 in Section 2.1.3 provides more detail about these messages. The *RRM Spare_Capacity_Report* is sent by the BS to ASN-GW to indicate its available radio resources [11]. We hypothesized that when configuring the BCR system parameters for the BS, they may get stored on the ASN-GW and passed between the ASN-GW and the BS. If this was true, it would allow for a possible man-in-the-middle attack to modify the packets that passed the BCR system parameters.

Through collaboration with the ORBIT management team we captured traffic on the Ethernet link between the ASN-GW and BS. Originally we followed the capturing process indicated in Figure 6.1. After review of this traffic, we were not able to find the system parameters *backoff start* and *backoff end*. With further thought, we captured the traffic again using the process depicted in Figure 6.3 in case these system parameters were passed between the ASN-GW and BS directly after a user set them. A small number of the packets captured between the ASN-GW and BS can be seen in Figure 6.4. After comparing the packets captured for various configurations of *backoff start* and *backoff end* we determined that they are not passed between the ASN-GW and BS.

Another R6 protocol message that was investigated was the *Path_Reg_Req* and *Path_Reg_Rsp*. These messages are used to request and establish the R6 data path between the ASN-GW and BS. The details of *Path_Reg_Rsp* can be seen in Figure 6.5. Similar to the ping reply seen in Section 6.2, the first information seen in the packet is the Ethernet destination and source MAC addresses followed by the IP address information. Following the IP information is the WiMAX ASN control path information. There is a variety of information TLV encoded in this part of the packet. We see the BSID near the beginning of this part of the packet. The TLV encodings do not follow the R6 standard due to the proprietary R6+ protocol being used.

6.4 Summary

Network traffic captures are common tool for reverse engineering communication protocols and gathering information. In this chapter, the traffic captured at two very diverse WiMAX related interfaces are discussed. These two interfaces are the **wmx0** interface, acts as the network device interface for the Linux OS, and the Ethernet interface between the WiMAX BS and ASN-GW.

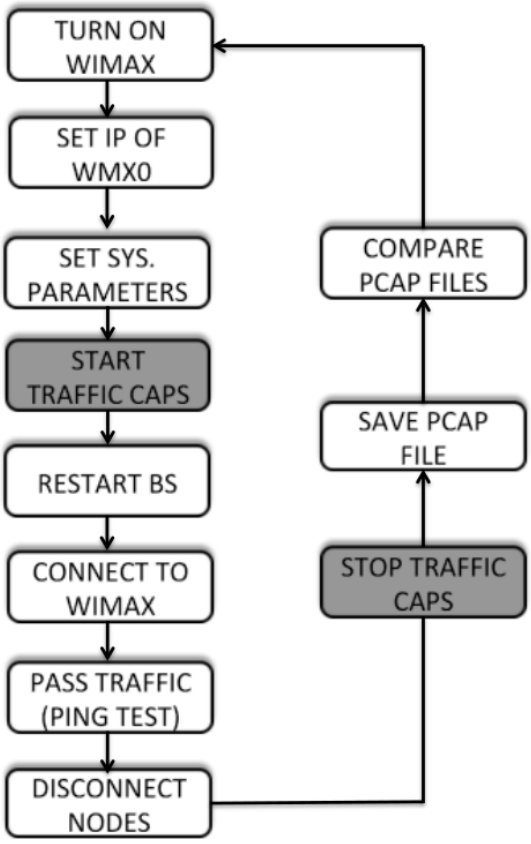

Figure 6.3: Flow chart of traffic capturing process on the R6+ interface between the ASN-GW and BS

30	11.109384	10.3.0.61	10.3.0.71	WiMAX	142 Context_Ack[Malformed Packet]
31	11.192608	10.3.0.61	10.3.0.71	WiMAX	206 Context_Rpt[Malformed Packet]
32	11.193156	10.3.0.71	10.3.0.61	WiMAX	210 Context_Req[Malformed Packet]
33	11.194406	10.3.0.61	10.3.0.71	WiMAX	142 Context_Ack[Malformed Packet]
34	11.197170	10.3.0.71	10.3.0.61	WiMAX	474 Path_Reg_Rsp[Malformed Packet]
35	11.284019	10.3.0.61	10.3.0.71	WiMAX	226 Path_Reg_Ack[Malformed Packet]
36	11.466764	10.3.0.71	10.3.0.61	WiMAX	122 Path_Reg_Req[Malformed Packet]
37	11.500989	10.3.0.61	10.3.0.71	WiMAX	150 R4/R6 Spare_Capacity_Rpt[Malformed Packet]
38	12.000987	10.3.0.61	10.3.0.71	WiMAX	150 R4/R6 Spare_Capacity_Rpt[Malformed Packet]
39	12.501027	10.3.0.61	10.3.0.71	WiMAX	150 R4/R6 Spare_Capacity_Rpt[Malformed Packet]

Figure 6.4: Traffic capture on the R6+ interface between the ASN-GW and BS

The protocol used on the latter interface connection is a proprietary protocol, R6+, that differentiates from the standard control protocol, R6, for management communication between the BS and ASN-GW. Two packets, a ping reply and PATH_REG_RSP, are parsed and the various protocol encapsulation is revealed. Unfortunately, we were not able to capture MAC management messages at the **wmx0** interface providing more evidence that these control and management messages along with the BCR system parameters are handled and isolated in the firmware on the WiMAX devices. Additionally, the few R6 protocol messages that may have contained the system parameters were

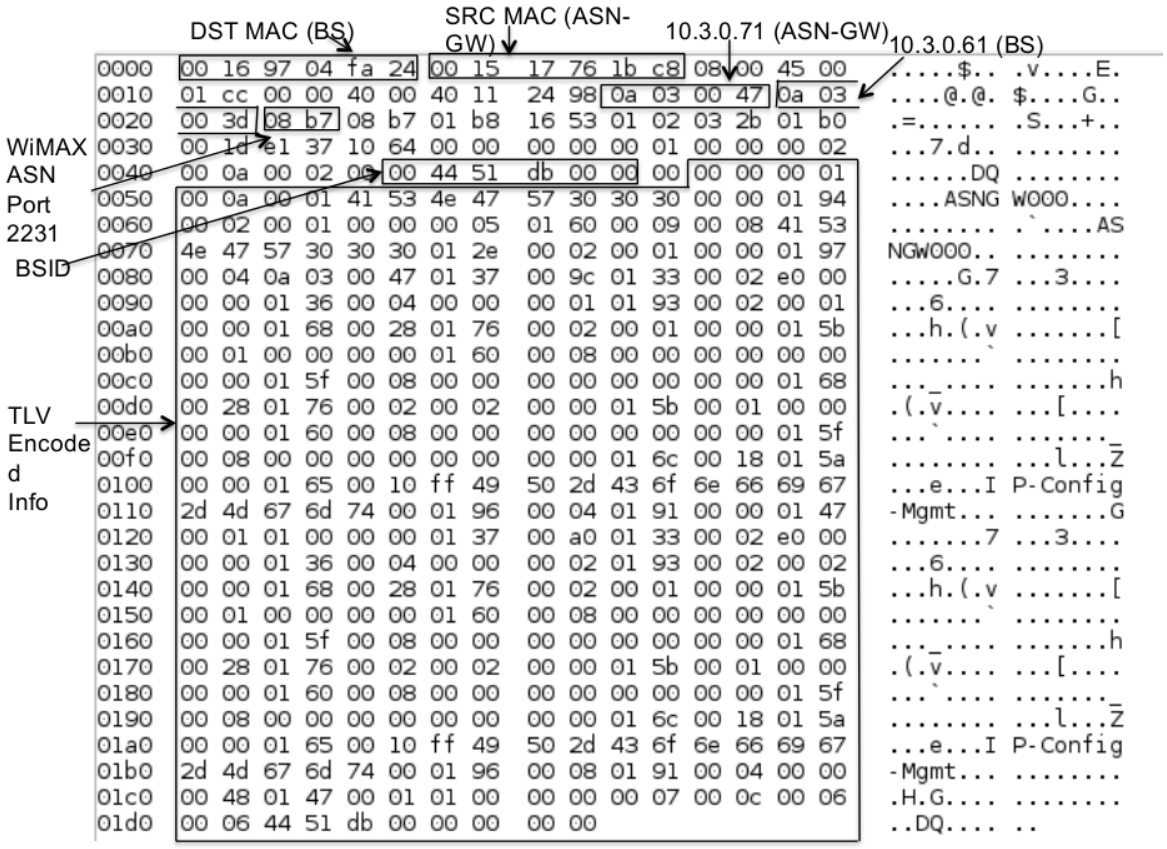

Figure 6.5: *Path_Reg_Rsp message details*

not discovered in the R6+ protocol used in this test environment. The following chapter will draw conclusion about this study and present the current status of this work.

Chapter 7

Conclusion

7.1 Summary

Wireless technologies are now a common part of everyday communications. From past wireless protocols and wireless security research, we are aware that there are numerous security flaws apparent in wireless protocols and many of these flaws enable DoS attacks on the protocol. One novice DoS attack is presented in this study. Additionally, network and wireless security research is often conducted via software simulations and the results are applied to actual real-life, hardware networks without further investigation. Many times hardware experimentation does not produce the same results as software. We have found that stringent network and wireless security research usually requires two things: reverse engineering and hardware experimentation.

Our proposed DoS attack manipulates the Bandwidth Contention Resolution system parameters of the IEEE 802.16e protocol. Our study of this attack has resulted in reverse engineering the WiMAX protocol implemented in hardware and analysis of hardware WiMAX environments. This paper documents this process and can be used as a starting point for conducting similar investigation of wireless and network protocols. Also, details of the IEEE 802.16e standard and protocol, previous WiMAX security research, and NS-2 software simulations of the BCR system parameters affects on throughput and packet-loss are presented and discussed in this document. The initial purpose of this study was to replicate the NS-2 software simulations. Hardware experiments that manipulate the BCR system parameters has proven to be more challenging than the software simulations. Despite the obstacles we have faced in attempting hardware experiments, a myriad of informative informa-

tion related to WiMAX, Linux, and different wireless network environments have been discovered through this study.

Chapter 3 analyzed the indoor and outdoor WiMAX testbed environments of ORBIT. The results of this analysis suggest the importance of software simulations and hardware experimentation in network security testing. The indoor environment supports ideal conditions such as many software simulations do. Our study suggests there can be dramatic differences in the results of studies carried out in this environment compared to actual outdoor wireless networks. Chapter 4 and 5 present a variety of configurable and managed parameters that could be used in future WiMAX security research.

Finally, Chapter 6 looked at a WiMAX control protocol not commonly known, R6 protocol, and the WiMAX interface of the Linux operating system. This last chapter highlighted a different aspect of wireless networks that many of us do not consider and, in doing so, raised awareness of another piece of the network that is potentially vulnerable to attack. It also provided a better understanding as to how the Linux's network interface for WiMAX interacts with the actual WiMAX device. Though we hoped otherwise, the Linux WiMAX interface is basically an Ethernet interface. It does not handle any of the WiMAX layer messages.

Despite that we were unable to carry out our original goal of manipulating the BCR system parameters, this investigation has served to be very beneficial, and will hopefully provide guidance and aid to other conducting WiMAX and other network protocol hardware experiments . Unfortunately we are unable to draw firm conclusions about the BCR system parameters settings. Software simulations suggest that the client SS's settings of the parameters is critical for decreasing vulnerability to BCR system parameter DoS attack. Section 2.3.3 recommends setting levels for some of the system parameters to provide robustness to this type of DoS attack. The hardware experiments suggest that these settings may not be as critical in real networks. Without replicating attackers in our hardware experiments, we can not conclude for sure that this statement is true. Before bringing the document to a close, we will look at other work related to this study in Section 7.2 and, finally, work still needed to be completed for this investigation and possibly work that this study has discovered in Section 7.3.

7.2 NetKarma

It is always a pleasure to have ones work inspire work by others. The work presented in this study has already served to promote and further other research. A project initially began to capture and aid GENI research collaborated with us during our study because of our GENI experiments. NetKarma is a project of Indiana Universitys Data to Insight Center. The purpose is to capture and visualize provenance of GENI experiments.

The ns-2 software simulations from [7] was one of the first GENI related experiments visualized using this tool. We collaborated with the researchers at Indiana University to aid in this effort by helping them to understand the ns-2 simulations and data and what were the most important measurements. NetKarma captures not only provenance of packet movement, but also infers critical provenance regarding packets that were dropped, and by doing so is able to convey information about DDoS attacks through visualization that was done earlier through ANOVA [38]. This was accomplished by using an NS-2 extension that generates an XML and trace file. The two files are input to NetKarma and Cytoscape [5] is used to visualize this data.

Figure 7.1 [3] visualizes the packet-loss for a specific parameter configuration that varies *bw_ request_retry*. Each line that is seen in the visualization connects from the BS, at the origin, out to either a client or attacker node. Attacker nodes are the red lines and client nodes are yellow. Each row of figures represents a different value of *bw_request_retry*. The left most column represents the nodes that experience packet loss in red. Int the middle column, the size of the nodes is proportionate to the number of packets dropped. So the red nodes seen in the left column have the largest radius in the middle column. On the other hand, the right most column the size of the nodes are proportionate to the number of packets transmitted and successfully received.Table 7.1 summarize the parameter configuration for this visualization. The results of the visualization agree with the conclusions drawn by Deng [7]. The number of packets dropped significantly decreases when *bw_request_retry* is increased from 2 to 6.

Table 7.1: Experiment runs from ns-2 simulations used by NetKarma for Figure 7.1

Run ID	Frame Duration	# of Attackers/Clients	Dos_backoff_start	Dos_request_retry	Bw_backoff_start	Bw_request_retry
406	0.01	80/20	1	2	1	2
407	0.01	80/20	1	2	1	6
408	0.01	80/20	1	2	1	10

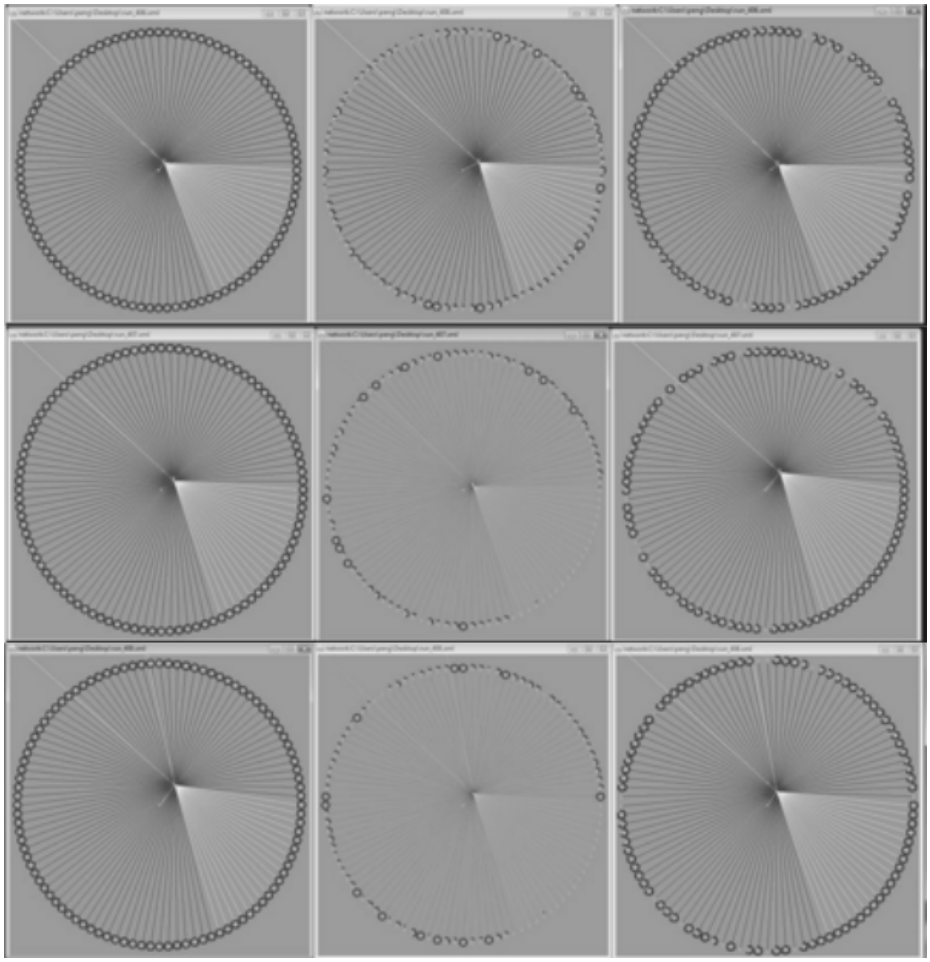

Figure 7.1: NetKarma visualization of *bw_request_retry* from software simulations

7.3 Future Work

There is a variety of current and future work related to this study. We are still pursuing some options for possibly controlling the BCR system parameters in ORBIT's WiMAX testbeds. The first option takes advantage of the indoor testbed and the UCD, UL-MAP interval. It would involve setting the BCR system parameters twice and using the indoor testbed attenuators to fade the path of specific nodes after the first set of system parameters. This would ideally allow for one set of nodes to use the first set of parameters and a second set of nodes to use the second configuration of system parameters. If possible, the fade on the first set of nodes would be removed bringing all nodes back to operating state on the network, and there would be a maximum of 10 seconds before the next UCD message would reset all system parameters to the same setting. This small interval

would allow for short experiments to be run. There are some logistics that currently have led us to doubt the plausibility of this technique. The biggest concern is that ORBIT's BS must be reset to take new BCR system parameter settings. Further investigation as to whether or not there is a way to set them on-the-fly is required.

The second option we may look at involves the BS to ASN-GW connection again. ORBIT, as well as Clemson University, now has an Airspan BS installed. The Airspan uses the standard R6 protocol for communication between the BS and ASN-GW. This opens new possibilities for a man-in-the-middle attack on this link. Also worth noting, is the manipulation of BCR system parameters can be accomplished on the physical layer if the right equipment was available. Packet construction is simple enough and the needed equipment would include Digital RF generator and wave generator at the appropriate frequencies. Unfortunately, this type of equipment was not available for this study.

Additionally, this study has opened many avenues for new future work. Due to the reverse engineering of WiMAX in Linux and the Intel® Centrino® Advanced + Wireless 6250 USB devices, there is now a clear understanding of the parameters that are handled outside of the firmware and by Linux operating system. Manipulation of these parameters would provide for interesting wireless security experiments and research. For example, manipulation of the QoS parameters could likely have negative effects on other SS on a network. Hopefully, this study will aid others in WiMAX hardware research and serve as informative resource about IEEE 802.16e, WiMAX, and reverse engineering network protocols.

Appendices

Appendix A MAC Layer Management Messages

MAC MANAGEMENT TYPE	MESSAGE NAME	DESCRIPTION	CONNECTION
0	UCD	Uplink Channel Descriptor	Broadcast
1	DCD	Downlink Channel Descriptor	Broadcast
2	DL_MAP	Downlink Access Definition	Broadcast
3	UL_MAP	Uplink Access Definition	Broadcast
4	RNG_REQ	Ranging Request	Broadcast
5	RNG_RSP	Ranging Response	Basic/Initial Ranging
6	REG-REQ	Registration Request	Basic/Initial Ranging
7	REG-RSP	Registration Response	Primary
8	reserved		Primary
9	PKM-REQ	Privacy Key Management Request	Primary
10	PKM-RSP	Privacy Key Management Response	Primary
11	DSA-REQ	Dynamic Service Addition Request	Primary
12	DSA-RSP	Dynamic Service Addition Response	Primary
13	DSA-ACK	Dynamic Service Addition Acknowledge	Primary
14	DSC-REQ	Dynamic Service Change Request	Primary
15	DSC-RSP	Dynamic Service Change Response	Primary
16	DSC-ACK	Dynamic Service Change Acknowledge	Primary
17	DSD-REQ	Dynamic Service Deletion Request	Primary
18	DSD-RSP	Dynamic Service Deletion Response	Primary
19	reserved		Primary
20	reserved		Primary
21	MCA-REQ	Multicast Assignment Request	Primary
22	MCA-RSP	Multicast Assignment Response	Primary
23	DBPC-REQ	Downlink Burst Profile Change Request	Basic
24	DBPC-RSP	Downlink Burst Profile Change Response	Basic
25	RES-CMD	Reset Command	Basic
26	SBC-REQ	SS Basic Capability Request	Basic
27	SBC-RSP	SS Basic Capability Response	Basic
28	CLK-CMP	SS network Clock Comparison	Broadcast
29	DREG-CMD	De/Re-register Command	Basic
30	DSX-RVE	DSx Received Message	Primary

MAC MANAGEMENT TYPE	MESSAGE NAME	DESCRIPTION	CONNECTION
31	TFTP-CPLT	Configuration File TFTP Complete Message	Primary
32	TFTP_RSP	Configuration File TFTP Complete Response	Primary
33	ARQ-Feedback	Standalone ARQ Feedback	Basic
34	ARQ-Discard	ARQ Discard message	Basic
35	ARQ-Reset	ARQ Reset message	Basic
36	REP-REQ	Channel measurement Report Request	Basic
37	REP-RSP	Channel measurement Report Response	Basic
38	FPC	Fast Power Control	Broadcast
39	MSH-NCFG	Mesh Network Configuration	Broadcast
40	MSH-NENT	Mesh Network Entry	Basic
41	MSH-DSCH	Mesh Distributes Schedule	Broadcast
42	MSH_CSCH	Mesh Centralized Schedule	Broadcast
43	MSH_CSCF	Mesh Centralized Schedule Configuration	Broadcast
44	AAS_FBCK_REQ	AAS Feedback Request	Basic
45	AAS_FBCK_RSP	AAS Reedback Response	Basic
46	AAS_Beam_Select	AAS Beam Select message	Basic
47	AAS_BEAM_REQ	AAS Beam Request message	Basic
48	AAS_BEAM_RSP	AAS Beam Response message	Basic
49	DREG_REQ	SS De-registration Request message	Basic
50	MOB_SLP_REQ	Mobile Sleep Request	Basic
51	MOB_SLP_RSP	Mobile Sleep Response	Basic
52	MOB_TRF_IND	Mobile Traffic Indication	Broadcast
53	MOB_NBR_ADV	Mobile Neighbor Advertisement	Broadcast/Primary
54	MOB_SCN_REQ	Mobile Scanning intevbal allocation Request	Basic
55	MOB_SCN_RSP	Mobile Scanning intevbal allocation Response	Basic
56	MOB_BSHO_REQ	BS Handover Request	Basic
57	MOB_MSHO_REQ	MS Handover Request	Basic
58	MOB_BSHO_RSP	BS Handover Response	Basic
59	MOB_HO_IND	Handover Indication	Basic
60	MOB_SCN_REP	Mobile Scanning result Report	Primary
61	MOB_PAG_ADV	BS broadcast Paging	Broadcast
62	MBS_MAP	MBS Map	Multibroadcast
63	PMC_REQ	Power control Mode Change Request	Basic
64	PMC_RSP	Power control Mode Change Response	Basic
65	PRC-LT-CTRL	Set-up/tear-down of Long-Term MIMO precoding	Basic
66	MOB_ASC-REP	Association result Report	Primary
67-255	Reserved		

Appendix B ORBIT Outdoor and Indoor Experiment Data

OUTDOOR											
	AVERAGE THROUGHPUT									SYS. PARAMETER	
RUN	NODE 1-1	NODE 1-2	NODE 1-3	NODE 1-4	NODE 1-5	NODE 1-6	NODE 1-7	NODE 1-8	SINK	BACKOFF_S	BACKOFF_E
1	1655.46667	1766.4	2048	2048	2048	2048	1962.66667	1954.13333	15513.6	3	2
2	1817.6	2048	2048	2048	2048	2048	1211.7333	2005.3333	15146.6667	1	10
3	2048	2030.9333	2048	2048	2048	2048	2030.9333	2048	16332.8	3	2
4	1877.3333	1979.73333	2048	2048	2048	2048	1783.46667	2048	15872	1	6
5	2056.3333	2056.3333	2056.3333	2048	2048	2039.46667	1902.93333	1928.53333	16102.4	1	10
6	2048	1996.8	2056.3333	2048	2048	2048	1459.2	2005.3333	15675.73	5	6
7	2039.46667	2005.3333	2056.3333	2048	2048	2048	1979.73333	1740.8	15948.8	5	2
8	2048	2048	2056.3333	2048	1681.06667	2048	2039.46667	1971.2	15914.67	1	6
9	2056.3333	2056.3333	2056.3333	2048	1638.4	2048	1911.46667	1954.1333	15744	3	10
10	2056.3333	2048	2048	2048	1996.8	2048	1945.6	1962.66667	16136.53	5	6
11	2022.4	2056.5333	2048	2048	2048	2048	1954.1333	2048	16264.53	3	2
12	2030.9333	2048	2048	2048	2048	2048	1979.73333	2048	16298.67	1	10
13	2030.9333	2030.933	2048	2048	2048	2048	1211.7333	2048	15496.53	5	10
14	2056.533	2056.5333	2056.5333	2048	2048	2039.46667	1979.73333	1629.86667	15880.53	3	6
15	2056.5333	2056.5333	2048	2048	2048	2048	2048	2048	16375.4667	5	10
16	2056.5333	2056.5333	2048	2048	2048	2048	2048	2048	16375.4667	5	6
17	2056.5333	2056.5333	2056.5333	2048	2048	2048	2048	2039.46667	16375.4667	3	10
18	2056.5333	2056.5333	2048	2048	2048	2048	2039.46667	2048	16366.9333	1	2
19	2056.5333	2056.5333	2048	2048	2048	2048	2048	2048	16384	5	2
20	2056.5333	2056.5333	2048	2048	2048	2048	2048	2048	16384	1	6
21	2056.5333	2056.5333	2056.5333	2048	2048	2048	2048	2048	16375.4667	5	10
22	2056.5333	2056.5333	2056.5333	2048	2048	2048	2048	2039.46667	16384	5	2
23	2056.5333	2056.5333	2056.5333	2048	2048	2048	2039.46667	2039.46667	16384	1	2
24	2056.5333	2056.5333	2048	2048	2048	2048	2048	2048	16401.0667	3	6
25	2056.5333	2056.5333	2048	2048	2048	2048	2048	2048	16384	3	10
26	2056.5333	2056.5333	2056.5333	2048	2048	2039.46667	2048	2048	16375.4667	3	6
AVG:	**2020.08**	**2034.86**	**2051.57**	**2048**	**2016.16**	**2047.02**	**1919.67**	**1997.78**	**16111.3**		

OUTDOOR

	PACKET LOSS									**SYS. PARAMETER**	
RUN	NODE 1-1	NODE 1-2	NODE 1-3	NODE 1-4	NODE 1-5	NODE 1-6	NODE 1-7	NODE 1-8	SINK	BACKOFF_S	BACKOFF_E
1	401.066667	290.133333		0	0	0	85.3333333	93.8666667	878.933333	3	2
2	238.933333	8.53333333	8.53333333	0	0	0	836.266667	42.6666667	1126.4	1	10
3	8.53333333	25.6	0	0	0	0	17.0666667	0	59.7333333	3	2
4	179.2	76.8	8.53333333	0	0	0	264.533333	0	529.066667	1	6
5	0	0	8.53333333	8.53333333	0	8.53333333	145.066667	119.466667	281.6	1	10
6	8.53333333	59.7333333	0	0	0	0	588.8	42.6666667	708.266667	5	6
7	17.0666667	51.2	0	8.53333333	0	0	68.2666667	307.2	452.266667	5	2
8	8.53333333	8.53333333	0	8.53333333	366.933333	0	8.53333333	76.8	477.866667	1	6
9	0	0	0	0	409.6	0	136.533333	93.8666667	640	3	10
10	0	8.53333333	0	8.53333333	51.2	0	102.4	85.3333333	264.533333	5	6
11	34.1333333	0	8.53333333	0	0	0	93.8666667	0	136.533333	3	2
12	25.6	8.53333333	8.53333333	0	0	0	68.2666667	0	110.933333	1	10
13	25.6	25.6	8.53333333	0	0	0	836.266667	0	896	5	10
14	0	0	8.53333333	0	0	8.53333333	68.2666667	418.133333	503.466667	3	6
15	0	0	8.53333333	0	0	0	0	0	8.53333333	5	10
16	0	0	8.53333333	0	0	0	0	0	8.53333333	5	6
17	0	0	0	8.53333333	0	0	0	8.53333333	17.0666667	3	10
18	0	0	8.53333333	0	0	0	8.53333333	0	17.0666667	1	2
19	0	0	8.53333333	0	0	0	0	0	8.53333333	5	2
20	0	0	8.53333333	0	0	0	0	0	8.53333333	1	6
21	0	0	0	0	0	0	0	8.53333333	8.53333333	5	10
22	0	0	0	0	0	0	0	8.53333333	8.53333333	5	2
23	0	0	0	0	0	0	17.0666667	8.53333333	25.6	1	2
24	0	0	8.53333333	0	0	0	0	0	8.53333333	3	6
25	0	0	8.53333333	0	0	0	0	0	8.53333333	3	10
26	0	0	0	0	0	8.53333333	0	0	8.53333333	3	6
AVG:	**67.6571**	**40.2286**	**4.59487**	**2.4381**	**59.1238**	**1.21905**	**237.105**	**91.4286**	**504.686**		

INDOOR											
	AVERAGE THROUGHPUT									SYS. PARAMETER	
RUN	NODE 1-1	NODE 1-2	NODE 1-3	NODE 1-4	NODE 1-5	NODE 1-6	NODE 1-7	NODE 1-8	SINK	BACKOFF_S	BACKOFF_E
1	2013.86667	2013.86667	2005.3333	2005.3333	2005.3333	2005.3333	1996.8	1996.8	16088.9492	3	2
2	2056.5333	2048	2048	2048	2048	2048	2048	2048	16384	1	10
3	2013.86667	2005.3333	2013.86667	2005.3333	2013.86667	2005.3333	1996.8	1996.8	16106.3051	3	2
4	1996.8	2005.3333	1988.26667	1996.8	1996.8	1988.26667	1971.2	1971.2	15958.7797	1	6
5	2005.3333	1988.26667	1979.73333	1988.26667	1988.26667	1988.26667	1971.2	1971.2	15941.4237	1	10
6	2013.86667	2005.3333	2005.3333	1996.8	2013.86667	2005.3333	1996.8	1996.8	16088.9492	5	6
7	2056.53333	2056.53333	2048	2048	2039.46667	2048	2048	2048	16653.0169	1	6
8	2056.5333	2048	2048	2048	2048	2039.46667	2039.46667	2039.46667	16401.3556	5	2
9	2065.06667	2065.06667	2056.3333	2056.3333	2048	2048	2048	2039.46667	16375.322	3	10
10	2005.3333	1988.26667	1996.8	1996.8	2005.3333	1996.8	1996.8	1996.8	16036.8814	5	6
11	2005.3333	1996.8	1996.8	2005.3333	1996.8	1996.8	1996.8	1996.8	16054.2374	3	2
12	2005.3333	2005.3333	2005.3333	2005.3333	2005.3333	2005.3333	2005.3333	2005.3333	16097.6271	1	10
13	2005.3333	2013.8667	2013.8667	2005.3333	2013.8667	2005.3333	2005.3333	2005.3333	16114.9831	5	10
14	2013.867	2013.8667	2005.3333	2013.8667	2005.3333	2005.3333	2005.3333	2005.3333	16132.339	3	6
15	2056.5333	2056.5333	2048	2048	2048	2048	2048	2048	16384	5	10
16	2056.5333	2056.5333	2056.5333	2048	2048	2048	2048	2048	16401.0667	5	6
17	2056.5333	2056.5333	2056.5333	2048	2039.46667	2048	2048	2048	16375.4667	3	10
18	2056.5333	2056.5333	2048	2048	2048	2048	2048	2048	16384	1	2
19	2056.5333	2056.5333	2048	2048	2048	2048	2048	2048	16384	5	2
20	2056.5333	2056.5333	2048	2048	2048	2048	2048	2048	16401.0667	1	6
21	2056.5333	2056.5333	2056.5333	2048	2048	2048	2048	2048	16384	5	10
22	2056.5333	2056.5333	2048	2048	2048	2039.46667	2048	2048	16375.4667	5	2
23	2056.5333	2056.5333	2048	2048	2048	2048	2048	2048	16392.5333	1	2
24	2056.5333	2056.5333	2048	2048	2048	2048	2048	2048	16384	3	6
25	2056.5333	2056.5333	2056.5333	2048	2048	2048	2048	2048	16384	3	10
26	2056.5333	2056.5333	2056.5333	2048	2048	2048	2048	2048	16384	3	6
AVG:	**2038.15**	**2035.86**	**2031.91**	**2030.6**	**2030.61**	**2028.96**	**2026.99**	**2026.67**	**16271.8**		

INDOOR											
	PACKET LOSS									SYS. PARAMETER	
RUN	NODE 1-1	NODE 1-2	NODE 1-3	NODE 1-4	NODE 1-5	NODE 1-6	NODE 1-7	NODE 1-8	SINK	BACKOFF_S	BACKOFF_E
1	72.81778	34.1333333	42.6666667	42.6666667	42.6666667	42.6666667	51.2	51.2	341.333333	3	2
2	0	8.53333333	0	0	0	0	0	0	8.53333333	1	10
3	34.1333333	42.6666667	34.1333333	42.6666667	34.1333333	42.6666667	51.2	51.2	332.8	3	2
4	51.2	42.6666667	59.7333333	51.2	51.2	59.7333333	76.8	76.8	469.333333	1	6
5	42.6666667	59.7333333	68.2666667	59.7333333	59.7333333	59.7333333	76.8	76.8	503.466667	1	10
6	34.1333333	42.6666667	42.6666667	51.2	34.1333333	42.6666667	51.2	51.2	349.866667	5	6
7	0	0	8.53333333	0	8.53333333	0	0	0	17.0666667	1	6
8	0	0	0	0	0	8.53333333	8.53333333	8.53333333	221.866667	5	2
9	0	0	8.53333333	8.53333333	17.0666667	17.0666667	17.0666667	25.6	93.8666667	3	10
10	42.6666667	59.7333333	51.2	51.2	42.6666667	51.2	51.2	51.2	401.066667	5	6
11	42.6666667	51.2	51.2	42.6666667	51.2	51.2	51.2	51.2	392.533333	3	2
12	42.6666667	42.6666667	42.6666667	42.6666667	42.6666667	42.6666667	42.6666667	42.6666667	341.333333	1	10
13	42.6666667	34.1333333	34.1333333	42.6666667	34.1333333	42.6666667	42.6666667	42.6666667	315.733333	5	10
14	34.1333333	34.1333333	42.6666667	34.1333333	42.6666667	42.6666667	42.6666667	42.6666667	315.733333	3	6
15	0	0	8.53333333	0	0	0	0	0	8.53333333	5	10
16	0	0	0	0	0	0	0	0	0	5	6
17	0	0	0	8.53333333	8.53333333	0	0	0	17.0666667	3	10
18	0	0	8.53333333	0	0	0	0	0	8.53333333	1	2
19	0	0	8.53333333	0	0	0	0	0	8.53333333	5	2
20	0	0	8.53333333	0	0	0	0	0	8.53333333	1	6
21	0	0	0	0	0	0	0	0	0	5	10
22	0	0	8.53333333	0	0	8.53333333	0	0	17.0666667	5	2
23	0	0	8.53333333	0	0	0	0	0	8.53333333	1	2
24	0	0	8.53333333	0	0	0	0	0	8.53333333	3	6
25	0	0	0	0	0	0	0	0	0	3	10
26	0	0	0	8.53333333	0	0	0	0	8.53333333	3	6
AVG:	**16.9135**	**17.3949**	**21.0051**	**18.7077**	**18.0513**	**19.6923**	**21.6615**	**21.9897**	**161.477**		

Appendix C Cirriculum Vitae

KATHERINE CAMERON

Phone: 503-516-0492

Email: kccamer@clemson.edu

15 Riggs Hall, Clemson University

Clemson, SC 29634

Expertise: Network and computer security, intelligent systems, communications, and embedded systems

ACADEMIC

M.S. Electrical Engineering May 2013
Clemson University, Clemson, SC GPA: 3.73/4.00

B.S. Electrical Engineering May 2008
University of Portland, Portland, OR GPA: 3.52/4.00

CAREER HISTORY

Clemson University, ECE Department Clemson, SC 08/11 Present
Research Assistant

- Conducting research under Dr. Richard R. Brooks in computer and network security.
- Projects include WiMAX vulnerability to DDoS attacks and deception as computer security counter measure.
- Conducted research independently, analyzed data, and documented and published the research methods and results.

Bonneville Power Administration Salem, OR/ Vancouver, WA 07/08 08/11
Field Communications Engineer

- Second in command for assurance and maintenance of communication system for our district and management responsibilities when supervisor was absent.
- Installed and integrated telemetry and communication equipment for new Waste Management generation site, designed and implemented digital communication system to complete Phase II SONET ring and construct un-collapsed fibre ring, upgraded communication systems from DS1 to DS3 capacity for video conferencing purposes.
- Managed craftsman and contractors, applied troubleshooting skills to electronic equipment under urgent deadlines, updated district documentation resulting in work efficiency.

Telecom Design Engineering Aid 05/07 07/08

- Communication site design and development.

- Evaluated new sites and designed tower, antenna, and battery systems for SONET ring construction, analyzed microwave paths for radio hops of SONET ring, oversaw turn-up and testing of radios and multiplex equipment for SONET ring.
- Documented designs, ordered materials, and coordination with other disciplines and contractors.

RESEARCH EXPERIENCE

EAGER-GENI Network Security and Traffic Analysis Clemson, SC 08/11 Present
Clemson University, Dr. Richard R. Brooks

- Investigation of the effect of IEEE 802.16e protocol Bandwidth Contention Resolution system parameters on vulnerability to Distributed Denial of Service (DoS) attacks.
- Applied factorial experiment design and ANOVA analysis.
- Simulation of experiments on remote hardware and presentation of work at multiple conferences.

Deception Countermeasures for Software Attacks Clemson, SC 08/12 01/13
Clemson University, Dr. Richard R. Brooks, and Sentar

- Developed, modeled, and analyzed innovative countermeasures to attacks on critical software. Import controlled project.

PUBLICATIONS AND PRESENTATIONS

- R. R. Brooks, "Introduction to Computer and Network Security: Navigating Shades of Gray", CRC Press, Boca Raton, FLA, (2013)
- ACM Conference on Cyber Security and Information Intelligence: Publication, Cost-Effective Quality Assurance of Wireless Network Security (October 2012)
- GENI Engineering Conference 14: Poster presentation, Status Quo: WiMAX Parameters and MAC-Level DoS Attacks Boston, MA (July 2012).
- N2 Womens Workshop: Poster presentation, Effects of WiMAX Parameters on MAC-Level DoS Attacks, Orlando, FL (March 2012).
- GENI Research and Experimenter Education, GREE 12: Publication, WiMAX: Bandwidth Contention Resolution, Vulnerability to Denial of Service Attacks (March 2012)
- GENI Engineering Conference 13: Poster presentation, Effects of WiMAX Parameters on MAC-Level DoS Attacks Los Angles, CA (March 2012)

Bibliography

[1] Derrick D Boom. Denial of service vulnerabilities ieee 802.16 wireless networks. Technical report, DTIC Document, 2004.

[2] Andries Brouwer. The linux kernel, 2003.

[3] Peng Chen. Role of provenance in visualizing packet throughput and packet loss. 07/2012 2012.

[4] P. Chi and C. Lei. A prevention approach to scrambling attacks in wimax networks. In *In World of Wireless, Mobile and Multimedia Networks & Workshop*, pages 1–8, 06/2009 2009.

[5] Cytoscape Consortium. Cytoscape. http://www.cytoscape.org/, 2013.

[6] NEC Corporation. Nec access service network. http://www.nec.com/en/global/solutions/nsp/WiMAX/products/access.html, 2009.

[7] Juan Deng, Richard R Brooks, and James Martin. Assessing the effect of wimax system parameter settings on mac-level local dos vulnerability. *International Journal of Performability Engineering*, 8(2):183, 2012.

[8] WiMAX Forum. Mobile wimax - part 1: Technical overview and performance evaluation. 08/2006 2006.

[9] WiMAX Forum. Wimax forum network architecture. stage 2: Architecture tenets, reference model and reference points part 1 release 1.0 version 4. wmf-t32-002-r010v04. 02/2009 2009.

[10] WiMAX Forum. Wimax forum network architecture: Detailed protocols and procedures base specification. wmf-t33-001-r015v02. 11/2010 2010.

[11] WiMAX Forum. Wimax forum network architecture stage 3: Detailed protocols and procedures. wmf-t33-001-r010v05. 02/2010 2010.

[12] WIreshark Foundation. Tshark. http://www.wireshark.org/docs/man-pages/tshark.html, 2011.

[13] WIreshark Foundation. Wireshark. http://www.wireshark.org/, 2011.

[14] Lee Garber. Denial-of-service attacks rip the internet. *IEEE Computer*, 33(4):12–17, 2000.

[15] Machtelt Garrels. *Introduction to Linux*. Fultus Publishing, 2010.

[16] IEEE 802.16 Working Group et al. Ieee 802.16–2004 ieee standard for local and metropolitan area networks part 16: Air interface for fixed and mobile broadband wireless access systems. *standards. ieee. org/getieee802/download/802.16 e-2004. pdf*, 2004.

[17] IEEE 802.16 Working Group et al. Ieee 802.16 e–2005 ieee standard for local and metropolitan area networks part 16: Air interface for fixed broadband wireless access systems amendment for physical and medium access control layers for combined fixed and mobile operation in licensed bands. *standards. ieee. org/getieee802/download/802.16 e-2005. pdf*, 2005.

[18] J Han, Mohamad Yusoff Alias, and Goi Bok Min. Potential denial of service attacks in ieee802. 16e-2005 networks. In *Communications and Information Technology, 2009. ISCIT 2009. 9th International Symposium on*, pages 1207–1212. IEEE, 2009.

[19] J. Hong, M. Alias, and B. Goi. Simulating denial of service attack using wimax experimental setup. *International Journal of Network and Mobile Technologies*, 2-1:30–34, 01/2011 2011.

[20] Junbeom Hur, Hyeongseop Shim, Pyung Kim, Hyunsoo Yoon, and Nah-Oak Song. Security considerations for handover schemes in mobile wimax networks. In *Wireless Communications and Networking Conference, 2008. WCNC 2008. IEEE*, pages 2531–2536. IEEE, 2008.

[21] Alefiya Hussain, John Heidemann, and Christos Papadopoulos. A framework for classifying denial of service attacks. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 99–110. ACM, 2003.

[22] Information Sciences Institute. The newtork simulator. http://www.isi.edu/nsnam/ns/, 2011.

[23] ITU-T International Telecommunication Union. X.690 series x: Data networks and open system communications, osi networking and system aspects- abstract syntax notation one (asn.1). 07/2002 2002.

[24] Internet2. Internet2. http://www.internet2.edu/, 2012.

[25] Houda Labiod, Afifi Hossam, and Costantino De Santis. Wi-fi, bluetooth, zigbee and wimax. 2007.

[26] Pero Latkoski, Zoran Hadzi-Velkov, and Borislav Popovski. Modeling and optimization of bandwidth request procedure in ieee 802.16 networks. In *Personal Indoor and Mobile Radio Communications (PIMRC), 2010 IEEE 21st International Symposium on*, pages 1469–1474. IEEE, 2010.

[27] Felix Lau, Stuart H Rubin, Michael H Smith, and Ljiljana Trajkovic. Distributed denial of service attacks. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 3, pages 2275–2280. IEEE, 2000.

[28] 2008 Intel Corporation ¡linux wimax@intel.com¿. Driver for the intel wireless wimax connection 2400m. 2008.

[29] 2008 Intel Corporation ¡linux wimax@intel.com¿. Linux kernel wimax stack. 2008.

[30] Douglas C Montgomery. *Design and analysis of experiments*. Wiley, 1991.

[31] Sheraz Naseer, Muhammad Younus, and Attiq Ahmed. Vulnerabilities exposing ieee 802.16 e networks to dos attacks: A survey. In *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2008. SNPD'08. Ninth ACIS International Conference on*, pages 344–349. IEEE, 2008.

[32] Qiang Ni, Alexey Vinel, Yang Xiao, Andrey Turlikov, and Tao Jiang. Wireless broadband access: Wimax and beyond-investigation of bandwidth request mechanisms under point-to-multipoint mode of wimax networks. *Communications Magazine, IEEE*, 45(5):132–138, 2007.

[33] NICTA. Omf unlock your experiments. http://www.mytestbed.net/, 2012.

[34] NLR. The network for advanced research and innovation. http://www.nlr.net/, 2005.

[35] Loutfi Nuaymi. *WiMAX: technology for broadband wireless access*. Wiley, 2007.

[36] Sanida Omerovic. Wimax overview. *Faculty of Electrical Engineering, University of Ljubljana, Slovenia*, 2006.

[37] Konstantinos Pelechrinis, Marios Iliofotou, and Srikanth V Krishnamurthy. Denial of service attacks in wireless networks: The case of jammers. *Communications Surveys & Tutorials, IEEE*, 13(2):245–257, 2011.

[38] Beth Plale, Peng Chen, Devarshi Ghoshal, and Yuan Luo. Visualization of network data provenance. 11/2012 2012.

[39] Ramjee Prasad and Fernando J Velez. *WiMAX Networks: Techno-Economic Vision and Challenges*. Springer, 2010.

[40] Dipankar Raychaudhuri, Ivan Seskar, Maximilian Ott, Sachin Ganu, Kishore Ramachandran, Haris Kremo, Robert Siracusa, Hang Liu, and Manpreet Singh. Overview of the orbit radio grid testbed for evaluation of next-generation wireless network protocols. In *Wireless Communications and Networking Conference, 2005 IEEE*, volume 3, pages 1664–1669. IEEE, 2005.

[41] Alessandro Rubini and Jonathan Corbet. *Linux device drivers*. O'reilly, 2001.

[42] Andrew Rutherford. *Introducing ANOVA and ANCOVA: a GLM approach*. Sage Publications Limited, 2001.

[43] Alexander Sayenko, Olli Alanen, and Timo Hämäläinen. Scheduling solution for the ieee 802.16 base station. *Computer Networks*, 52(1):96–115, 2008.

[44] Jolyon White, Guillaume Jourjon, Thierry Rakotoarivelo, and Max Ott. Measurement architectures for network experiments with disconnected mobile nodes. In *Proc. of TridentCom*, volume 46, pages 350–365, 2010.

[45] Rutgers University's WINLAB. Orbit lab. http://www.orbit-lab.org/, 2011.